Reaction Based Grasp Force Assignment

DISSERTATION

Mark W. Hunter
Captain, USAF

AFIT/DS/ENY/96-10

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/DS/ENY/96-10

Reaction Based Grasp Force Assignment

DISSERTATION

Mark W. Hunter
Captain, USAF

AFIT/DS/ENY/96-10

AFIT/DS/ENY/96-10

Reaction Based Grasp Force Assignment

DISSERTATION

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

Mark W. Hunter, B.S., M.S.
Captain, USAF

Sponsored by SA-ALC/RACE

~~September~~ July, 1996

AFIT/DS/ENY/96-10

Reaction Based Grasp Force Assignment

Mark W. Hunter, B.S., M.S.

Captain, USAF

Approved:

Curtis B. Spanny	3 July 1996
Dan L. Schrieber	3 July 1996
Brad S. Liebst	3 JULY 1996
[illegible]	3 JULY 1996
Kuldip S Rattan	9 July 1996

Robert A. Calico, Jr.

Dean

Acknowledgements

It is difficult to concisely express my appreciation to all those who have contributed to the research embodied in this document. However, I'd like to attempt to thank those individuals who have made this work possible.

First, I'd like to thank my research advisor Dr. Curtis Spenny, to whom I am most indebted. Dr. Spenny has always been willing to discuss problems, offer suggestions, and challenge my solutions. Under his guidance I have gained greater appreciation of the subtleties of the problems I have attempted to solve and solutions I have attempted to understand. His vision of a functional robotic manipulation scheme helped focus my research and provided the context for it's use.

I would like to acknowledge the members of my research committee, which provided both breadth and depth of expertise and enhanced this research. Dr. Kuldip Rattan's knowledge of both robotics and fuzzy logic was crucial to this research. He provided important insight to both the theory and implementation of fuzzy logic systems. Dr. Rattan was always enthusiastic and patient throughout this work and is deeply appreciated. I wish to thank Dr. Brad Liebst for his keen review of my work, as well as his probing questions, which has made this research stronger. I also wish to thank Major Dean Schneider whose enthusiastic and unwavering support strengthened me in moments of doubt and helped me through the CHIMERA learning curve. I'd also like to thank my Dean's representative, Lieutenant Colonel Terry Caipen for his technical review of the manuscript. The committee's insightful comments and suggestions helped me to better express my ideas and results, which significantly improved the quality of this dissertation.

I'd also like to thank Mr. Steve Parmley, without whom I may never have gotten my C-code to compile or work properly in the CHIMERA environment. His patience, skill, and humor was most appreciated. I'd like to thank Mr. Dave Doak who was always willing to see me through my numerous conflicts with the UNIX operating system. I wish to thank Mr. Dan Zambon, Ms. Kristin Larsen, and Mr. Mark Derriso for the computer and experimental support I needed. Additionally, I'd like to thank Captain Rich Cobb, who has preceded me in this academic endeavor, for his support concerning MATLAB, LaTeX, and a myriad of other details which had to be sorted out. These people made this research experience interesting and fun and enabled me to complete it with a minimum of hair loss.

Most of all, I'd like to thank my wife Toni, my daughter Victoria, and my mom Joann. Even before I began the research, they had no doubt I would be successful. They gave me the strength and desire to finish the task. I love them all dearly.

Mark W. Hunter

Table of Contents

List of Figures

List of Symbols

a ..... acceleration

$\mathbf{A}$ ..... matrix of orthonormals of the null space of $\mathbf{W}$

$\mathcal{A}$ ..... fuzzy set

a_c ..... magnitude of largest contact disturbance

a_m ..... magnitude of largest contact disturbance as function of $\mathbf{f}_i$

abs ..... absolute value

$\mathbf{B}$ ..... frictional constraint matrix

$\mathbf{B}_g$ ..... grasping force mapping matrix

$\mathbf{B}_m$ ..... manipulative force mapping matrix

β ..... friction parameter

$\mathbf{C}$ ..... manipulator Coriolis and centrifugal matrix

dx ..... x-coordinate of closest approach to friction cone

$\mathbf{D}$ ..... manipulator inertial matrix

$D_{\mathcal{N}}$ ..... dimension of null space

$\frac{d}{dt}$ ..... derivative with respect to time

$\mathbf{\Delta}\hat{\mathbf{f}}(t)$ ..... iterative change in internal force

$\mathbf{\Delta}\hat{\mathbf{F}}(t)$ ..... vector of iterative change in internal forces

δx ..... convergence cone offset distance

$\mathbf{e}$ ..... unit vector

$\mathbf{e\_I}$ ..... internal force unit vector

$\bar{\mathbf{e}}$ ..... $\mathbf{B}_m$ basis vector

$\mathbf{er\_}\hat{\mathbf{f}}(t)$ ..... iterative error in contact force

er_dot ..... dot product of contact force error and internal force

$\mathbf{E}$ ..... identity matrix

$\mathbf{ER\_DOT}$ ..... vector of er_dot values

ε ..... augmented objective function

$\mathbf{f}$ ..... contact force

$\bar{\mathbf{f}}$ ..... actual applied contact force

$\hat{\mathbf{f}}(t)$ ..... iterative contact force

$\mathbf{f}_{obj}$ ..... sum of forces about the object

ϕ ..... object coordinates

$\phi(t)$ ..... angle of $\hat{\mathbf{f}}(t)$ in local y-z plane

$\mathbf{F}$ ..... vector of contact forces

$\mathbf{f}int$ ..... internal contact force

$\mathbf{f}ext$ ..... external contact force

$\mathbf{F}int$ ..... vector of internal contact forces

$\mathbf{F}ext$ ..... vector of external contact forces

$\mathbf{g}$ ..... gravity vector

g ..... constraint equation

h ..... objective function

$\mathbf{h}_g$ ..... vector of grasp parameters

$\mathbf{h}_m$ ..... vector of manipulation parameters

$\mathbf{I}$ ..... object moment of inertia matrix

$\mathbf{I}_{obj}$ ..... augmented object moment of inertia matrix

$\mathbf{J}$ ..... manipulator Jacobian matrix

$\mathbf{K}$ ..... constant coefficient matrix

$k(t)$ ..... internal force scale factor

$\bar{k}(t)$ ..... alternative internal force scale factor

k_1, k_2, k_3 ..... manipulative force mapping matrix parameters

L ..... contact controlling scale factor solution

L_N ..... N-valued Logic

Λ ..... vector of Lagrange multipliers

λ ..... Lagrange multiplier

m ..... number of contacts

Max ..... algebraic equivalent to logical max

max ..... logical maximum value of a set

Min ..... algebraic equivalent to logical min

min ..... logical minimum value of a set

m_{obj} ..... mass of object

μ ..... coefficient of static friction

$\mathbf{n}_{obj}$ ..... sum of moments about the object

$\eta_{\mathcal{A}}(x)$ ..... membership function over the domain $\mathcal{A}$

N_I ..... number of internal forces

N_{F^2} ..... number of independent terms in $\|\mathbf{F}\|^2$

O ..... coordinate system origin

$\mathbf{O}$ ..... zero matrix

ω ..... angular velocity

$\mathbf{p}$ ..... position vector with respect to object frame

$\mathbf{P}$ ..... mapping operator from force to moment

$\mathbf{Pos}$ ..... matrix of position vectors in column order

$\mathbf{Q}$ ..... object wrench

$\mathbf{Q}_{obj}$ ..... object wrench due to gravity

$\mathbf{r}$ ..... position vector with respect to inertial frame

$\Re$ ..... real numbers

S ..... t-conorm

t ..... iteration

T ..... t-norm

τ ..... generalized manipulator force vector

θ ..... generalized manipulator position vector

U ..... fuzzy domain

v ..... velocity

$\mathbf{W}$ ..... grasp matrix

$w(t)$ ..... change in internal force weight

$\mathbf{x}_i$ ..... i^{th} contact local basis vector

X ..... fuzzy domain

$\mathbf{y}$ ..... vector of internal force magnitudes

$\mathbf{y}_i$ ..... i^{th} contact local basis vector

$\mathbf{z}_i$ ..... i^{th} contact local basis vector

Superscripts

$^{-1}$ ..... inverse

$^{\#}$ ..... pseudoinverse

^{o} ..... minimizing values

^{T} ..... transpose

Subscripts

$_{1}$ ..... 1^{st} component

$_{2}$ ..... 2^{d} component

$_{3}$ ..... 3^{d} component

$_{CMD}$ ..... commanded

$_{d}$ ..... desired

$_{f}$ ..... due to contact force

$_{g}$ ..... grasping

$_{i}$ ..... i^{th} contact

$_{ij}$ ..... from i^{th} to j^{th} contact

$_{m}$ ..... manipulating

$_{n}$ ..... normal

$_{N}$ ..... normal

$_{o}$ ..... inertial

$_{obj}$ ..... object

$_{p}$ ..... due to manipulator

$_{t}$ ..... tangential

$_{x}$ ..... x-component

$_{y}$ ..... y-component

$_{z}$ ..... z-component

Other

$\dot{}$ ..... first derivative with respect to time

$\ddot{}$ ..... second derivative with respect to time

$\| \|$ ..... Euclidean norm

$\forall$ ..... for all

Abstract

An iterative method is developed by which the contact forces required to apply an arbitrary wrench (six elements of force and moment) to a stably grasped object may be calculated quickly. The assignment of contact forces, given a required object wrench, is accomplished with the use of fuzzy logic. This concept is referred to as the fuzzy logic reactive system (FLRS). The solution is versatile with respect to goals inherent in the rulebase and the input parameters, and is also applicable for an arbitrary number of contacts. The goal presented in this research, to illustrate the concept of the FLRS, is the minimization of the norm of the contact forces using point contacts with friction. Results comparing the contact force assignment for this method and the optimal method proposed by Nakamura are presented. The results show that FLRS will satisfy the object wrench and frictional contacts while achieving near optimal contact force assignment. This method is shown to require significantly fewer floating point operations than the solution calculated using numerical constrained optimization techniques.

Reaction Based Grasp Force Assignment

I. Introduction

1.1 Motivation

The word *robot* was introduced to the world through a satirical drama written in 1921 by the Czech playwright Karl Capek [26]. The robots were depicted as anthropomorphic machines which excelled in the physically demanding work environment of the early twentieth century. This idealistic vision of machines, equipped with human looking dextrous hands, has yet to be realized. This is due in part to the complex nature of object manipulation by multifingered dextrous hands.

Object manipulation is a common problem for robots, prosthetics, and artificially stimulated biological hands [14, 40, 36, 33, 23]. The recent past is filled with various control architectures, grasp philosophies, and numerical methods which have yet to equal the human ability to stably manipulate grasped objects [4, 37, 43]. Grasped object manipulation has been, and continues to be, an active area of research with many papers produced for robotics journals, conferences, and symposiums [60, 25, 9]. The goal of this research is to contribute to the body of knowledge of real-time solution techniques, concerning the problem of stable manipulation of grasped objects.

Many robots today are limited to environments where the robot tools are essentially special end-effectors locked in place which may or may not be difficult to change. In any case, the tools are of limited use and of special design consistent with structured use of the robot. This lack of tool generality limits the ability of a robot to function in unstructured environments. This in turn limits the argument for

robots in general, since hard automation machines may be more suited for exclusively repetitive tasks in a predictable environment.

The manipulation of an object by humans is greatly enhanced by the usually seamless integration of sensors, actuators, and controllers acting on multiple levels. The technological challenge facing engineers and scientists today, related to grasping and manipulation, is the development of integrated artificial "hands" as capable as those of humans. Such integration involves low level manipulative elements as well as higher level planning elements. The lower level elements are responsible for the mechanics of basic object manipulation. One of the elements necessary to successfully control a redundantly grasped object is a fast contact force assignment method based on commanded object behavior. Many current methods emphasize solutions which are of limited application or numerically intensive and globally optimal. This work investigates a different approach to the solution of this problem.

1.2 Overview of Conventional Grasped Object Manipulation Methods

The literature defines several classes of robotic grasps. Napier suggests that human grasps are variations of either a power or precision grasp, we will examine the precision grasp only [41]. The precision grasp is one in which the grasped object is contacted by the fingertips only and is generally associated with fine manipulation under low force applications. Different forms of control architectures for the manipulation of grasped objects have been proposed. Nakamura, among others, separates the grasped object control problem into two parts, dynamic and static. The dynamic portion concerns the determination of joint torques required to accelerate the end-effectors of the finger manipulators to follow the motion of the contact points on the accelerated object. The static portion controls the joint torques required to satisfy contact stability and accelerate the object [39]. Schneider and Cannon use this architecture, but also include external forces applied to the object among static force components [49]. Schneider and Cannon describe a feedback system which is

an implementation of object based impedance control based on the work of Hogan [20]. Their control scheme enforces a controlled impedance of the object and thus environmental contacts are easily handled.

Li, Hsu, and Sastry have proposed a unified control scheme which seeks to control the manipulators and object together to yield a specified object position and internal force trajectory [32]. Yoshikawa et al. have proposed using forms of the so called hybrid position/force control; this method accomplishes force and position control along orthogonal directions [61, 44, 63]. Again this method presupposes the existence of a satisfactory internal force trajectory. Michelman and Allen have proposed and demonstrated an elaboration of the hybrid scheme with the use of a blend of predefined manipulative behaviors to accomplish a desired object manipulation [34]. The predefined manipulative behaviors enforce the partition of the task space into force or position controlled directions. However, this method also requires knowledge of the internal force requirements necessary to maintain contact stability while implementing the manipulation.

A common thread throughout the various control architectures is the requirement of *contact stability* and the need for commanded manipulator contact forces. A *stable contact* is one in which the applicable frictional constraints are satisfied. In other words,

$$\mathbf{f}_t \leq \mathbf{f}_n \cdot \mu \tag{1.1}$$

where $\mathbf{f}_n$ is the normal contact force component, μ is the coefficient of static friction, and $\mathbf{f}_t$ is the tangential force component, assuming a Coulomb friction model. The vector of generalized contact forces, $\mathbf{F}$, is related to the object *wrench*, $\mathbf{Q}$, where the *wrench* is a vector of object forces and moments at a specified point, through the grasp matrix [7, 38, 32, 25]. The grasp matrix, $\mathbf{W}$, maps contact forces to object *wrench*,

$$\mathbf{Q} = \mathbf{WF} \tag{1.2}$$

where $\mathbf{W}$ is a matrix containing elements of the contact positions, in the object frame. Full column rank of $\mathbf{W}$ implies the grasp is one of *force closure*. A force closure grasp is one in which any object wrench may be resisted by some combination of contact forces, i.e. a solution exists.

The kinematic solution commonly used to determine $\mathbf{F}$ decomposes contact forces into two orthogonal sets.

$$\mathbf{F} = \mathbf{F}ext + \mathbf{F}int \tag{1.3}$$

This is usually accomplished through the use of a generalized inverse of the grasp matrix $\mathbf{W}$. A common definition of *external* forces, $\mathbf{F}ext$: are those forces, the sum of which oppose the resultant wrench applied to the object in the course of accelerating the object or reacting forces applied to the object. *Internal* forces, $\mathbf{F}int$, are those forces, the sum of which produce no net wrench on the object but which serve to achieve contact stability [39, 25, 35]. Thus, the external forces are dictated by the desired object behavior, while the internal forces are dictated by the need to maintain stable contacts on the object. Yoshikawa et al. have defined *grasping* and *manipulative* forces similar to the internal and external forces described above [63].

Some authors propose using geometric properties of the grasp configuration to determine internal forces; these solutions are not general in nature [47, 63, 10, 21, 24]. Park and Starr have proposed calculating the internal force for a given grasp configuration and scaling this solution as necessary to satisfy contact stability [43]. Many authors refer to *optimal* contact forces. Some choose this to mean that the final contact force solution lies as far from the constraint boundaries as possible [25]. This definition of optimal solution suffers on two counts. First, since this solution entails having larger forces than those required for contact stability, power requirements for the manipulator will be higher than necessary. Second, small errors in the direction of the contact forces, introduced by geometric uncertainties, may generate relatively

large deviations from the desired object acceleration [15]. Nakamura has described this effect as similar to tightly squeezing a wet bar of soap [40]. Kumar and Waldron have proposed seeking solutions where all contact forces pass through a point of concurrence. This point is chosen to be the centroid of the contact points [30]. Orin and Oh have proposed a solution based on the minimization of the grasp power and normal force; by linearizing the constraints, they propose to use linear programming methods to solve the problem [42].

Nakamura, Yoshikawa, and other authors have defined optimal contact forces as those which seek to minimize the Euclidean norm of the contact forces while satisfying contact stability requirements [38, 63]. This objective function does not suffer from the previously mentioned problem of squeezing too tightly, though the frictional constraints must be shifted inward to allow some measure of solution robustness in the face of unknown disturbances. If one seeks the minimum Euclidean norm of the contact forces, the pseudoinverse or weighted pseudoinverse solution may be used for the calculation of external contact forces. This solution will provide the minimum Euclidean norm of the external forces. However, the solution will almost always violate the contact frictional constraints. The pseudoinverse solution has two parts: the actual pseudoinverse of the grasp coefficient matrix multiplied with the resultant force vector and the matrix, whose columns span the null space of the grasp coefficient matrix, multiplied with an arbitrary vector which then defines the internal force [43, 47]. It is the internal force components which must be added to the external force components to achieve a valid solution. The pseudoinverse solution tends to distribute the object wrench equally between the contacts; the internal forces (null space solution) modify the resultant force at each contact so as to satisfy friction constraints.

For gripped contacts (i.e. contacts with no frictional constraints), the pseudoinverse solution is sufficient for implementation of a manipulation scheme [3, 6, 56, 59]. However, contacts involving friction require further constraints on the solution.

Specifically, that the contact forces must lie in the friction cone. This additional constraint requires the solution of a non-linear programming problem, which can be accomplished in an iterative manner for planar problems, as proposed by some authors [38]. As mentioned earlier, the constraints on the Euclidean norm solution should be modified to generate some measure of contact stability [39]. This method is generally not satisfactory with regards to solution speed and has not been effectively demonstrated [37, 30].

Recently, Buss et al. have proposed that this problem be viewed as a linearly constrained semidefinite programming problem for which there are globally exponentially convergent solutions via gradient flows [9]. The algorithm input requires an initial solution satisfying the frictional constraints in order for the method to render a valid solution. The objective function is a mix of the sum of the contact normal forces and a quantity tending to infinity for contact forces on the edge of the friction cone. This method also incorporates weighting matrices which allow the qualitative weighting of the objective function. The method is reported to have real-time solution capabilities. Holzmann and McCarthy have also recently proposed computing the friction forces associated with a three-fingered grasp using a genetic algorithm [22]. Other recent papers include the impressive demonstration of a robust internal-force based impedance control method accomplished by Bonitz and Hsia, which also requires the specification of the internal forces required to maintain contact stability. [8]. Sinha and Abel have proposed using elastic modeling of the manipulator/object interface in order to avoid using the common Coulomb friction constraints [50].

1.3 Problem Statement

The focus of this research is the development of a real time capable, spatial contact force assignment method for determining the individual contact contributions, specifically the internal force contributions, for the manipulation of a redundantly grasped object. Fast solution speed is important since ideal internal force magnitudes

will vary with changing external load on the grasped object. Solutions qualitatively similar to optimal solutions based on the minimization of the Euclidean norm of the contact forces are sought. Additionally, the solution should be flexible with regards to robust contact stability requirements and allow implicit control of the available internal forces.

1.4 Proposed Solution

Some scientists have proposed a reaction based control scheme for basic artificial life forms [54]. This idea was born of the need to simplify the architecture of decision making for artificially intelligent agents. Rather than having a large centralized knowledge based inference engine forced to maintain a consistent logical model of the outside world, *subsumption* architecture relies on a decentralized reactionary control scheme with the intervention of higher level control in the event of conflicting lower level components. This architecture builds complex behaviors from conglomerations of far simpler ones.

The philosophy behind this research is the idea that perhaps a better way to solve the grasping problem is similar to the *subsumption* architecture idea as related to artificial intelligence; that is, to speed-up the problem solution by decentralizing the solution generation. In the past, authors have attempted to solve the contact force assignment problem while considering all contacts simultaneously, a computationally expensive proposition. This research will investigate the problem solution through pair wise analysis of the contact forces. Thus, using the *subsumption* analogy, the proposed method is *reactive.* This proposed method also shares a similar solution structure with the *subsumption* architecture, both methods lend themselves to parallel processing.

The proposed hand kinematic solution is for the spatial problem of grasping a rigid object by multiple finger contacts stably. The contacts are assumed to be frictional fingertip point contacts which can achieve arbitrary forces. Also, the contact

geometry, friction constraints, and commanded object wrench are assumed known. The solution involves the use of an iterative scheme which starts with the pseudoinverse solution of the contact forces. During each loop of the iterative portion, incremental changes in the internal forces will be made based on outputs from a non-linear inference process.

This inference will be accomplished through the use of a fuzzy logic system; though, as proved later, this is not required for solution convergence. The process developed to calculate the valid contact forces will be referred to as the fuzzy logic reactive system (FLRS). The fuzzy inference portion of FLRS is comprised of a knowledge-based system consisting of IF...THEN rules associated with vague predicates and a fuzzy logic inference mechanism. The definition of the input parameters and output values are based on the heuristic pair wise evaluation of the contact forces and how the shared internal force may be used to satisfy the contact constraints efficiently. Thus, instead of globally driving all contact forces to minimize some objective function, each internal force is evaluated separately and are then combined to form the overall incremental change in contact forces.

Once a basic system has been demonstrated, one may use one of many automated adaptive techniques to tune the parameters controlling the mapping to achieve better results [12, 16, 31]. An alternative approach would be to learn the FLRS parameters from sets of approved input-output data, much like the training of artificial neural networks [46, 55, 53].

1.5 Solution Validation

The FLRS system developed will be validated on several levels.

Criteria for solution convergence of force closure grasps (grasps with stable solutions to any commanded object wrench), will be shown for both the two and m-contact cases.

The FLRS algorithm will be compared to an exact solution as proposed by Nakamura for a two contact case. The FLRS method will also be compared with numeric approximations of the Nakamura method, which determines contact forces that satisfy the frictional constraints and minimize the Euclidean norm of the contact forces through an optimization algorithm. The examples will consist of various spatial grasp configurations with a range of commanded object wrenches. A comparison of the contact forces will be made along with the floating point operations required to calculate the solution. The numerical environment used for the comparison will be MATLAB. The specific numeric optimization algorithm used will be the *CONSTR* function. This function solves the constrained optimization problem, in Kuhn-Tucker form, using sequential quadratic programming methods.

A test of the FLRS real time capability will be accomplished be translating the algorithm to C and compiling the code for use by the Chimera real time operating system. The code will be executed on a Motorola 68030 microprocessor on a VMEbus card.

II. Background

2.1 The Multirobot Coordination Problem

As mentioned in the introduction, many authors have reported various levels of success in developing control algorithms to coordinate multiple robotic mechanisms. The following development of an overall control strategy is the concept proposed by Nakamura [40]. As mentioned in Chapter I, there are other architectures, each requiring contact forces which satisfy both contact stability and object manipulation requirements. The Nakamura architecture is representative and is presented here to define the context in which this research in contact force allocation can be used. This research is applicable to other architectures as well. Nakamura proposes an architecture which explicitly controls both the manipulator and the object as the basis for solving the general precision grasping and manipulation problem.

Using the notation of Nakamura, the following is a development for coordinative manipulation of a grasped object by m robotic mechanisms, as shown in Figure 2.1. The basic assumptions in this development are:

- Grasped object is rigid
- Position of the contacts on the object are known
- Tangent plane of the object surface at the points of contact are known
- Object mass is known
- Object velocity can be measured
- Contact configuration provides force closure grasp
- Manipulator end point makes point contact with the surface of the object
- End points do not move on or away from the surface of the object
- Gravity vector is known

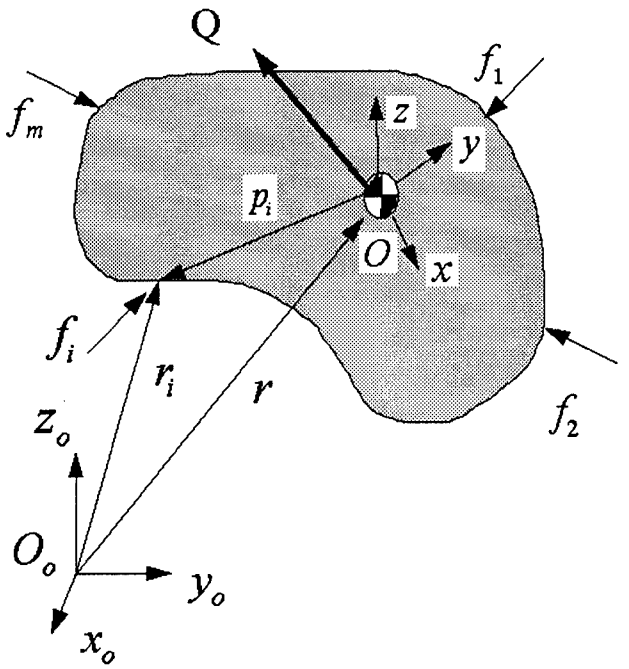

Figure 2.1 Model and nomenclature for rigid object grasped by m manipulators

These assumptions allow us to consider contacts which produce only forces on the grasped object and are capable of producing an arbitrary object *wrench*. The object wrench, $\mathbf{Q}$, at the object center of mass consists of forces, $\mathbf{f}_{obj}$, and moments, $\mathbf{n}_{obj}$, due to the contact forces and may be represented by:

$$\mathbf{f}_{obj} = \sum_{i=1}^{m} \mathbf{f}_i + m_{obj}\mathbf{g} \tag{2.1}$$

$$\mathbf{n}_{obj} = \sum_{i=1}^{m} (\mathbf{p}_i \times \mathbf{f}_i) \tag{2.2}$$

where $\mathbf{f}_i$ are the contact forces, m_{obj} is the mass of object, $\mathbf{g}$ is the gravitational acceleration, and $\mathbf{p}_i$ is the position of the i^{th} contact in the object frame. The equations of motion due to the contact forces may be represented by the Newton-Euler equations:

$$m_{obj}\ddot{\mathbf{r}} = \mathbf{f}_{obj} \tag{2.3}$$

$$\mathbf{I}\dot{\omega} + \omega \times (\mathbf{I}\omega) = \mathbf{n}_{obj} \tag{2.4}$$

where $\ddot{\mathbf{r}}$ is the second derivative of the position vector of the object with respect to some inertial frame, $\mathbf{I}$ is the moment of inertia matrix, and ω is the rotational velocity vector.

The above equations are used to formulate the equations of motion into the compact form below:

$$\mathbf{I}_{obj}\ddot{\phi} + \mathbf{Q}_{obj} = \mathbf{Q} \tag{2.5}$$

where

$$\ddot{\phi} = \begin{pmatrix} \ddot{\mathbf{r}}^T & \dot{\omega}^T \end{pmatrix}^T \in \Re^6 \tag{2.6}$$

$$\phi = \begin{pmatrix} \mathbf{r}^T & \theta^T \end{pmatrix}^T \in \Re^6 \tag{2.7}$$

$$\mathbf{I}_{obj} = \begin{pmatrix} m_{obj}\mathbf{E}_3 & \mathbf{O} \\ \mathbf{O} & \mathbf{I} \end{pmatrix} \in \Re^{6\times 6} \tag{2.8}$$

$$\mathbf{Q}_{obj} = \begin{pmatrix} -m_{obj}\mathbf{g}^T & \omega \times (\mathbf{I}\omega)^T \end{pmatrix}^T \tag{2.9}$$

$$\mathbf{Q} = \mathbf{W}\mathbf{F} \in \Re^6 \tag{2.10}$$

$$\mathbf{F} = \begin{pmatrix} \mathbf{f}_1^T & \mathbf{f}_2^T & \cdots & \mathbf{f}_m^T \end{pmatrix}^T \in \Re^{3m} \tag{2.11}$$

$$\mathbf{f}_i = \begin{pmatrix} f_{i1} & f_{i2} & f_{i3} \end{pmatrix}^T \in \Re^3 \tag{2.12}$$

$$\mathbf{W} = \begin{pmatrix} \mathbf{E}_3 & \mathbf{E}_3 & \cdots & \mathbf{E}_3 \\ \mathbf{P}_1 & \mathbf{P}_2 & \cdots & \mathbf{P}_m \end{pmatrix} \in \Re^{6\times 3m} \tag{2.13}$$

$$\mathbf{P}_i = \begin{pmatrix} 0 & -p_{i3} & p_{i2} \\ p_{i3} & 0 & -p_{i1} \\ -p_{i2} & p_{i1} & 0 \end{pmatrix} \in \Re^{3\times 3} \tag{2.14}$$

$$\mathbf{p}_i = \begin{pmatrix} p_{i1} & p_{i2} & p_{i3} \end{pmatrix}^T \in \Re^3 \tag{2.15}$$

where $\mathbf{E}_n$ is the n^{th} order identity matrix and the position and force components 1, 2, and 3 are nominally the object x, y, and z axes respectively. We could have included an external applied force and moment in this development. Equations 2.1 through 2.5 would have reflected this change in the applied wrench; and would have been similar to the development accomplished by Schneider and Cannon [49]. However, for purposes of putting this research in context, that addition is not necessary.

If the desired object trajectory is given by $\phi_d(t) \in \Re^6$; then, if the net wrench

$$\mathbf{Q} = \mathbf{Q}_{obj} + \mathbf{I}_{obj}\{\ddot{\phi}_d + \mathbf{K}_1(\dot{\phi}_d - \dot{\phi}) + \mathbf{K}_2(\phi_d - \phi)\} \tag{2.16}$$

is applied to the grasped object, the object's motion will be governed by

$$(\ddot{\phi}_d - \ddot{\phi}) + \mathbf{K}_1(\dot{\phi}_d - \dot{\phi}) + \mathbf{K}_2(\phi_d - \phi) = 0 \tag{2.17}$$

where $\mathbf{K}_1$ and $\mathbf{K}_2$ are constant coefficient matrices chosen to guarantee asymptotic convergence and ϕ is the actual object trajectory. Equation 2.17 indicates that the actual object trajectory converges to the desired object trajectory. This behavior translates into dynamically stable response to commanded object trajectories and is referred to, by Nakamura, as *object stability* [40]. For this control scheme to be implemented, one must have ϕ, $\dot{\phi}$, and the contact forces necessary to generate the object wrench, $\mathbf{Q}$.

The dynamic portion of this control problem seeks to generate the generalized joint forces necessary to drive each manipulator through the commanded trajectory. Below, the position control algorithm is formulated. The acceleration of any contact point is:

$$\ddot{\mathbf{r}}_i = \ddot{\mathbf{r}} + \dot{\omega} \times \mathbf{p}_i + \omega \times (\omega \times \mathbf{p}_i) \tag{2.18}$$

where the terms $\ddot{\mathbf{r}}$ and $\dot{\omega}$ denote the object accelerations, both linear and rotational, resulting from the object wrench, $\mathbf{Q}$. These may also be represented as:

$$\begin{pmatrix} \ddot{\mathbf{r}}^T & \dot{\omega}^T \end{pmatrix} = \ddot{\phi} \tag{2.19}$$

$$= \ddot{\phi}_d + \mathbf{K}_1(\dot{\phi}_d - \dot{\phi}) + \mathbf{K}_2(\phi_d - \phi) \tag{2.20}$$

$$= \mathbf{I}_{obj}^{-T}(\mathbf{Q} - \mathbf{Q}_{obj}) \tag{2.21}$$

The i^{th} generalized manipulator joint force which results in end-effector $\ddot{\mathbf{r}}_i$, when the robot is not making contact with the object, is τ_{ip}. This value is the trajectory following generalized force associated with non-contact manipulators,

$$\tau_{ip} = \mathbf{D}_i(\theta_i)\ddot{\theta}_{iCMD} + \mathbf{C}_i(\theta_i, \dot{\theta}_i)\dot{\theta} + \mathbf{g}_i(\theta_i) \tag{2.22}$$

where

$$\ddot{\theta}_{iCMD} = \mathbf{J}_i(\theta_i)^{-1}\left(\ddot{\mathbf{r}}_i - \frac{d}{dt}\mathbf{J}_i(\theta_i)\dot{\theta}_i\right) \tag{2.23}$$

and $\mathbf{J}_i$, $\mathbf{D}_i$, $\mathbf{C}_i$, and $\mathbf{g}_i$ denote manipulator Jacobian, inertial, Coriolis and centrifugal, and gravity terms respectively for the i^{th} manipulator, in terms of the generalized manipulator coordinate vector θ_i. This relationship, when coupled with an error reducing feedback in position and velocity, is referred to as *resolved acceleration control* [17].

Now consider the generalized force required of the i^{th} manipulator to apply the end-effector force $\mathbf{f}_i$ statically to the grasped object. This generalized force is denoted τ_{if}. Compute τ_{if} by multiplying the transposed Jacobian matrix of the i^{th} manipulator by the i^{th} end-effector force $\mathbf{f}_i$. According to *d'Alembert's principle,* the net generalized force τ_i required to manipulate the object is the sum of τ_{ip} and τ_{if}, as shown in Figure 2.2 [40]. The symbol $\bar{\mathbf{f}}_i$ represents the actual contact force applied to the object by the i^{th} manipulator.

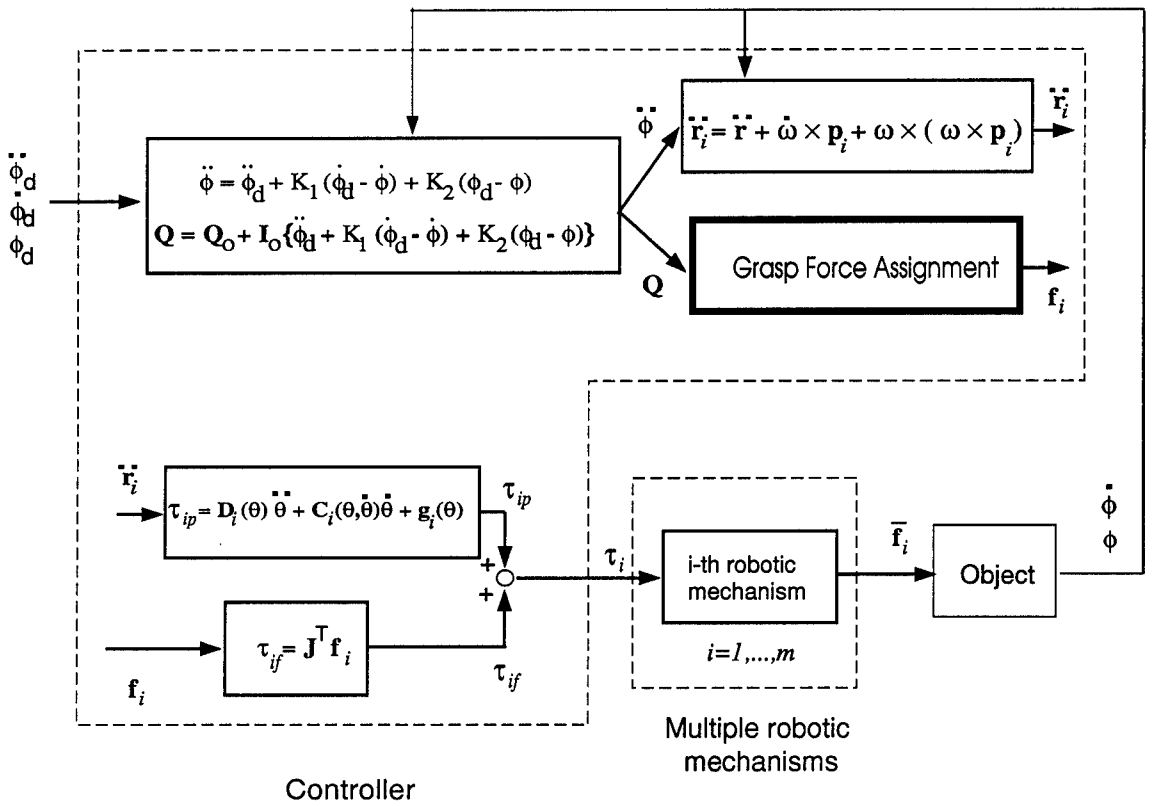

Figure 2.2 Feed forward portion of object manipulation controller

An important aspect of the control model developed are the assumptions that the contact forces commanded can in fact be delivered by the grasping robot manipulators and that the object position and velocity may be accurately determined. The challenges regarding the physical implementation of a robot capable of operating as modeled is not to be underestimated. Two key elements, of a physical implementation of a grasping robot, are the development of high fidelity fingertip force sensors and actuators.

2.2 Grasp Force Assignment

The "Grasp Force Assignment" block of Figure 2.2 is the keystone of the grasped object controller, this portion is solely responsible for the distribution of the contact forces and the maintenance of contact stability. This block is required to operate in real-time for hardware implementations of this architecture. Two methods commonly referred to for this purpose will be presented. The first method, proposed by Nakamura, attempts to find an optimal solution based on the minimum Euclidean norm of the contact forces. The second method, proposed by Nagai and Yoshikawa, places additional constraints on the same optimization solution in order to increase solution speed.

2.2.1 Nakamura's Solution. A common method, for the calculation of the finger contact forces necessary to generate a given object wrench, is the pseudoinverse solution which is exemplified by the methods proposed by Nakamura et al. [38]. The pseudoinverse is one of the infinite number of generalized inverse solutions for the finger contact force vector $\mathbf{F}$ of Equation 2.10. In general, the contact forces, $\mathbf{F}$, constitute a redundant set for the solution of the equation. Thus, a solution for the contact forces through a generalized inverse of the grasp matrix $\mathbf{W}$ will yield two parts, as in Equation 2.24.

$$\mathbf{F} = \mathbf{W}^{\#}\mathbf{Q} + \mathbf{A}\mathbf{y} \tag{2.24}$$

where $\mathbf{W}^{\#} \in \Re^{3m \times 6}$ is the pseudoinverse of the grasp matrix, $\mathbf{W}$. The term $\mathbf{A}$, is the matrix of the orthonormals of the null space of $\mathbf{W}$, and $\mathbf{y}$ is an arbitrary vector. Thus, $\mathbf{Ay}$ is an element of the null space of $\mathbf{W}$, and $D_{\mathcal{N}} = 3m - rank(\mathbf{W})$ is the dimension of the null space of $\mathbf{W}$. Many authors use the *Moore-Penrose inverse,* or pseudoinverse, solution as this solution has the property that $\mathbf{W}^{\#}\mathbf{Q}$ minimizes the Euclidean norm of the contact forces [40].

The term $\mathbf{W}^{\#}\mathbf{Q}$ is often referred to as the *external* force vector, $\mathbf{F}ext$, since this is the only term which contributes to the desired object wrench, $\mathbf{Q}$.

$$\mathbf{F}ext = \mathbf{W}^{\#}\mathbf{Q} \tag{2.25}$$

$$\mathbf{F}ext = \begin{pmatrix} \mathbf{f}ext_1^T & \mathbf{f}ext_2^T & \cdots & \mathbf{f}ext_m^T \end{pmatrix}^T \in \Re^{3m} \tag{2.26}$$

The other element of the solution, $\mathbf{Ay}$, cannot contribute to the net wrench and is thus referred to as the *internal* force vector, $\mathbf{F}int$.

$$\mathbf{F}int = \mathbf{Ay} \tag{2.27}$$

$$\mathbf{F}int = \begin{pmatrix} \mathbf{f}int_1^T & \mathbf{f}int_2^T & \cdots & \mathbf{f}int_m^T \end{pmatrix}^T \in \Re^{3m} \tag{2.28}$$

where

$$\mathbf{f}int_i = \sum_{j=1, j \neq i}^{m} \mathbf{f}int_{ij} \tag{2.29}$$

and $\mathbf{f}int_{ij}$ is the internal force from contact i to contact j. Figure 2.3 illustrates the difference between the internal and external forces. Internal force pairs lie on a line connecting any two contact points; thus their directions are dictated by the geometry of the contact points, $\mathbf{W}$. The external force directions are dictated by both the contact geometry, *and* the object wrench, $\mathbf{Q}$. As Figure 2.3 illustrates, the effect of the pseudoinverse solution, for determining $\mathbf{F}ext$, is to distribute the external contact forces so as to resist a given object wrench in a minimum norm

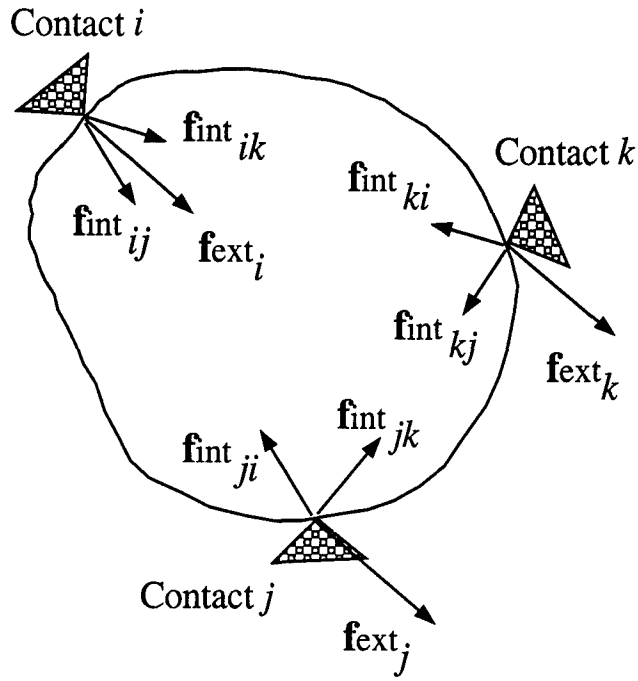

Figure 2.3 Internal and external forces

sense. For rigid contact grasp problems, where the contacts are fixed to the object, the $\mathbf{W}^{\#}\mathbf{Q}$ solution is entirely effective and reasonable. However, for contacts which rely on maintenance of contact forces that satisfy friction constraints, this solution will almost always violate frictional constraints, as illustrated by external contact forces $\mathbf{f}ext_k$ and $\mathbf{f}ext_j$, in Figure 2.3.

In order to satisfy the frictional constraints, illustrated in Figure 2.4, the solution derived from the pseudoinverse must be modified. The modifiers to the $\mathbf{W}^{\#}\mathbf{Q}$ solution are the internal forces. Nakamura determines these internal forces while continuing to minimize the Euclidean norm of the contact forces. The method outlined below is acceptable for solving the simple two contact problem but is prohibitively expensive with respect to computational time for more complex problems [37, 30].

Nakamura formulates and solves the constrained optimization problem using the Kuhn-Tucker theorem [40]. For the specified object wrench, $\mathbf{Q}$, obtain $\mathbf{F}$ that

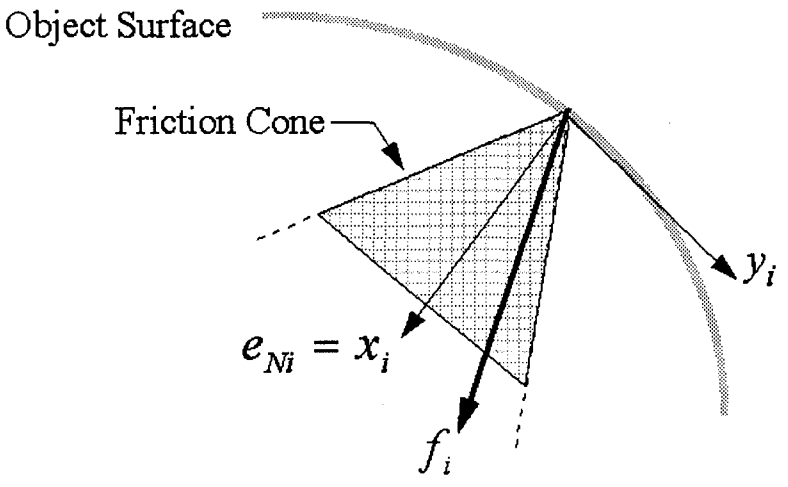

Figure 2.4 Contact friction constraint

satisfies

$$\min \|\mathbf{F}\| \tag{2.30}$$

with the constraints

$$\mathbf{Q} = \mathbf{WF} \tag{2.31}$$

$$\mathbf{e}_{Ni}^T \mathbf{f}_i \geq \beta_i \|\mathbf{f}_i\| \ \forall i \tag{2.32}$$

where $\beta_i = 1/\sqrt{1+\mu_i^2}$. We've previously assumed that the i^{th} finger is capable of applying an arbitrary force at the fingertip and that force closure exists. The constraints of Equation 2.32 may be written as:

$$g_i(\mathbf{y}) \leq 0 \quad i = 1, \ldots, 2m \tag{2.33}$$

where

$$g_i(\mathbf{y}) = \left\{ \begin{array}{ll} \mathbf{f}_i^T \mathbf{B}_i \mathbf{f}_i & i = 1, \ldots, m \\ -\mathbf{e}_{N(i-m)}^T \mathbf{f}_i & i = m+1, \ldots, 2m \end{array} \right\} \tag{2.34}$$

and $\mathbf{B}_i = \beta_i^2\mathbf{E}_3 - \mathbf{e}_{Ni}\mathbf{e}_{Ni}^T$. Since $\|\mathbf{F}\|$ is a positive function, we may declare an equivalent objective function to Equation 2.30 as:

$$\min h(\mathbf{y}) \tag{2.35}$$

where

$$h(\mathbf{y}) = \mathbf{F}^T\mathbf{F} \tag{2.36}$$

Thus the problem of determining the contact forces which minimize $\|\mathbf{F}\|$ has been reduced to minimizing a quadratic function with linear and quadratic inequality constraints. To determine the unknown variable $\mathbf{y}$, we apply the Kuhn-Tucker Theorem [40]. An augmented objective function, $\varepsilon(\mathbf{y}, \mathbf{\Lambda})$, is formed using the Lagrange multipliers $\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_{2m} \end{pmatrix}^T \in \Re^{2m}$,

$$\varepsilon(\mathbf{y}, \mathbf{\Lambda}) = h(\mathbf{y}) + \sum_{i=1}^{2m} \lambda_i g_i(\mathbf{y}) \tag{2.37}$$

where $\lambda_i \geq 0$. The Kuhn-Tucker Theorem, and the fact that the augmented objective function can be shown to be a convex function with respect to $\mathbf{y}$, leads to the necessary and sufficient conditions for the global minimum of $\varepsilon(\mathbf{y}, \mathbf{\Lambda})$, and thus $\|\mathbf{F}\|$ [40].

$$\left.\frac{\partial \varepsilon}{\partial \mathbf{y}}\right|_{\mathbf{y}=\mathbf{y}^o,\, \Lambda=\Lambda^o} = 0 \tag{2.38}$$

$$\sum_{i=1}^{2m} \lambda_i^o g_i(\mathbf{y}^o) = 0 \tag{2.39}$$

where $\mathbf{y}^o$ and $\mathbf{\Lambda}^o$ are the values of $\mathbf{y}$ and $\mathbf{\Lambda}$ which minimize the objective function and satisfy the constraints.

This formulation solves for all internal forces simultaneously which leads to extremely large polynomial objective functions for even modest contact configurations. The number of internal force pairs, N_I, to simultaneously specify for a solution is $(m^2-m)/2$. For a five contact problem, ten internal forces must be determined. Con-

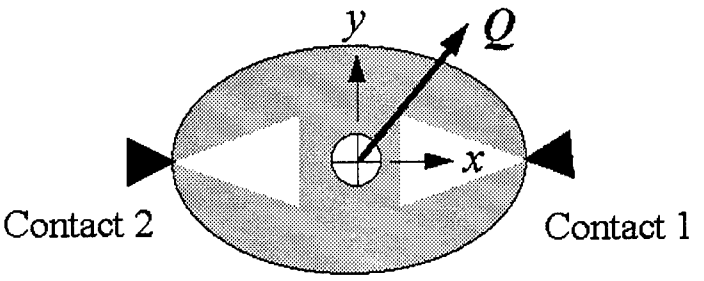

Figure 2.5 Two contact grasp example

sidering the case of five contacts on an object, $m = 5$, $\|\mathbf{F}_i\|^2$ is a 15 term quadratic equation in terms of the four relevant internal force magnitudes. The number of independent terms in $\|\mathbf{F}\|^2$ is $N_{F^2} = 1 + N_I + (N_I^2 - N_I)/2$, or 56 terms for the five contact problem. Simultaneously determining a minimum norm solution among all possible solutions is a daunting and time consuming task. Problems with more than two contacts are generally solved using numeric optimization methods. The statement of conditions for the global minimum allow numeric optimization algorithms to be guaranteed to converge to the global minimum.

2.2.2 Exact Solution Example . The exact solution using the previously outlined method can be determined for simple two contact problems. In this case the problem must be reduced to a planar one as the grasp matrix, $\mathbf{W}$, is not full rank for the spatial case and thus force closure will not exist. The elements of Equation 2.24 are thus reduced in dimension. The unknown variable y is a scalar for this case, indicating that we have only a single internal force with which to apply to the external force solution to enforce contact constraints.

Figure 2.5 illustrates the geometry, axis system, and commanded object wrench of a two contact example, in which the grasped object is an ellipse. The contact inward pointing normal directions are the negative of the respective position vectors. **Pos** is the matrix of position vectors, in column order.

$$\mathbf{Pos} = \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix} \tag{2.40}$$

Let the coefficient of static friction for both contacts, μ, be 0.4 and the commanded planar object wrench, $\mathbf{Q}$, be:

$$\mathbf{Q} = \begin{Bmatrix} 1 \\ 1 \\ 0 \end{Bmatrix} \tag{2.41}$$

The grasp matrix, $\mathbf{W}$, and the orthonormals of the null space of $\mathbf{W}$, $\mathbf{A}$, from Equation 2.40 are:

$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \quad \mathbf{A} = \begin{Bmatrix} -0.7071 \\ 0 \\ 0.7071 \\ 0 \end{Bmatrix} \tag{2.42}$$

The pseudoinverse of $\mathbf{W}$ is,

$$\mathbf{W}^{\#} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0 \\ 0 & 0.5 & -0.5 \end{bmatrix} \tag{2.43}$$

and Equation 2.24 may now be written as:

$$\mathbf{F} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0 \\ 0 & 0.5 & -0.5 \end{bmatrix} \begin{Bmatrix} 1 \\ 1 \\ 0 \end{Bmatrix} + \begin{Bmatrix} -0.7071 \\ 0 \\ 0.7071 \\ 0 \end{Bmatrix} y = \begin{Bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{Bmatrix} + \begin{Bmatrix} -0.7071 \\ 0 \\ 0.7071 \\ 0 \end{Bmatrix} y \tag{2.44}$$

Clearly, the sum of the external contact forces and moments equal the commanded object wrench.

By applying the only internal force available, the net contact forces will move inward toward the friction cones as the internal force magnitude, y, is increased. The contact force for the second contact, $\mathbf{f}_2$ will thus move inside the friction cone before $\mathbf{f}_1$. Therefore, the constraint equation $g_2(y) < 0$ and thus $\lambda_2 = 0$. We also know that, in general, both contacts will be exerting force. Therefore, $g_3(y) < 0$ and $g_4(y) < 0$ which requires $\lambda_3 = 0$ and $\lambda_4 = 0$. For this solution to be a minimum norm solution, one contact force must be on the friction cone, therefore $g_1(y) = 0$ Thus, we have reduced the five equations from the Kuhn-Tucker formulation to two equations, with two unknowns. y and λ_1.

The components of the augmented objective function of Equation 2.37 may now be formed.

$$h(y) = (0.5 - 0.7071y)^2 + 0.5^2 + (0.5 + 0.7071y)^2 + 0.5^2 \tag{2.45}$$

$$= 1 + y^2 \tag{2.46}$$

and

$$\mathbf{e}_{N1} = \begin{Bmatrix} -1 \\ 0 \end{Bmatrix} \quad \mathbf{B}_1 = \begin{bmatrix} \beta^2 - 1 & 0 \\ 0 & \beta^2 \end{bmatrix} \tag{2.47}$$

therefore

$$g_1(y) = (0.5 - 0.7071y)^2(\beta^2 - 1) + 0.5^2(\beta^2) \tag{2.48}$$

Equations 2.38 and 2.39 may now be constructed,

$$\left.\frac{\partial \varepsilon}{\partial y}\right|_{y=y^\circ, \Lambda=\Lambda^\circ} = 2y^\circ + 2\lambda_1^\circ \left[(-0.7071)(0.5 - 0.7071y^\circ)(\beta^2 - 1)\right] = 0 \tag{2.49}$$

$$\sum_{i=1}^{2m} \lambda_i^\circ g_i(y^\circ) = \lambda_1^\circ \left[(0.5 - 0.7071y^\circ)^2(\beta^2 - 1) + 0.25\beta^2\right] = 0 \tag{2.50}$$

and y^o may be solved for:

$$y^o = 0.7071 + \sqrt{.5 - \left(0.5\left(\frac{2\beta^2 - 1}{(\beta^2 - 1)}\right)\right)} \tag{2.51}$$

For this particular case, $y^o = 2.4744$, and the contact forces are:

$$\mathbf{F} = \begin{Bmatrix} -1.25 \\ 0.5 \\ 2.25 \\ 0.5 \end{Bmatrix} \tag{2.52}$$

For general and realistic grasping problems, at least one contact force will lie on the friction cone (necessary for a minimum norm solution), thus the exact solution to the two contact problem is relatively simple. Larger problems will in general exhibit coupling between Equations 2.38 and 2.39. Grasp configurations may occur, especially with large values of μ, in which the pseudoinverse solution does lie within all the contact constraints; one would then not need any additional contact force components.

2.2.3 Yoshikawa's Method of Contact Assignment. The normal component of the external force solution for Contact 1, of the two contact example above, was away from the surface. The external contact force solution in and of itself is clearly not sufficient for frictional contacts and is somewhat misleading in that some internal and external forces may be equal and opposite producing no net contact force. The method proposed by Yoshikawa and Nagai attempts to address this issue and to speed up the solution with additional constraints.

The force space is decomposed into manipulative and grasping forces, similar to the contact force development of Kobayashi [63, 27]. These forces are also similar to the internal and external force definitions described earlier. The grasping forces

are defined as those which lie in the null space of the grasp matrix $\mathbf{W}$ and satisfy the contact friction constraints. The manipulating forces are those which produce the desired resultant wrench, *do not act in the inverse direction to the grasping force, and are orthogonal to the grasping force components.* It is the additional constraints on the directions of the manipulating forces which differentiates this method from the method proposed by Nakamura et al. The method is applicable for two, three, and four contacts only. The contact forces are separated into two orthogonal sets:

$$\mathbf{F} = \mathbf{F}_g + \mathbf{F}_m \tag{2.53}$$

where $\mathbf{F}_g = \mathbf{B}_g\mathbf{h}_g$ and $\mathbf{F}_m = \mathbf{B}_m\mathbf{h}_m$. $\mathbf{B}_g \in \Re^{3m\times n}$, $\mathbf{h}_g \in \Re^n$, $\mathbf{B}_m \in \Re^{3m\times l}$, and $\mathbf{h}_m \in \Re^l$; where $n = \{1,3,6\}$ and $l = \{3,6,6\}$ for two, three, and four contact problems respectively. The unknown variables are the grasp parameters, $\mathbf{h}_g$.

The matrix $\mathbf{B}_g$ is determined from the contact geometry. For a three contact problem of a convex object,

$$\mathbf{B}_g = \begin{bmatrix} 0 & \mathbf{e}_{13} & \mathbf{e}_{12} \\ \mathbf{e}_{23} & 0 & \mathbf{e}_{21} \\ \mathbf{e}_{32} & \mathbf{e}_{31} & 0 \end{bmatrix} \tag{2.54}$$

The matrix $\mathbf{B}_m$ is a function of both the contact geometry *and* the commanded object wrench. Thus $\mathbf{B}_m$ must be calculated for each $\mathbf{Q}$.

$$\mathbf{B}_m = \begin{bmatrix} 0 & (1-k_2)\bar{\mathbf{e}}_{13} & k_3\bar{\mathbf{e}}_{12} & \bar{\mathbf{e}}_{10} & 0 & 0 \\ k_1\bar{\mathbf{e}}_{23} & 0 & (1-k_3)\bar{\mathbf{e}}_{21} & 0 & \bar{\mathbf{e}}_{20} & 0 \\ (1-k_1)\bar{\mathbf{e}}_{32} & k_2\bar{\mathbf{e}}_{31} & 0 & 0 & 0 & \bar{\mathbf{e}}_{30} \end{bmatrix} \tag{2.55}$$

where k_i are parameters which can take on the value of 0 or 1 and $\bar{\mathbf{e}}_{ij}$ are the basis vectors of the manipulating force (unit vectors from contact i to contact j).

$$\bar{\mathbf{e}}_{i0} = \frac{\mathbf{e}_{i(i+1)} \times \mathbf{e}_{i(i+2)}}{\left\| \mathbf{e}_{i(i+1)} \times \mathbf{e}_{i(i+2)} \right\|} \tag{2.56}$$

$$\bar{\mathbf{e}}_{i(i+1)} = \mathbf{e}_{i(i+2)} \times \bar{\mathbf{e}}_{i0} \tag{2.57}$$

$$\bar{\mathbf{e}}_{i(i+2)} = \bar{\mathbf{e}}_{i0} \times \mathbf{e}_{i(i+1)} \tag{2.58}$$

The parameters k_i are chosen such that:

$$\mathbf{h}_m = (\mathbf{W}\,\mathbf{B}_m)^{-1}\,\mathbf{Q} \tag{2.59}$$

where $h_{mi} \geq 0$.

The solution method proposed by the authors requires either a planned trajectory of grasp parameters, $\mathbf{h}_g$, which are known to satisfy the frictional constraints or an iterative solution of the unknown grasp parameters [37]. This method may be thought of as similar to the method of Nakamura with additional constraints associated with the direction of the orthogonal forces. However, the extra computations associated with $\mathbf{B}_m$ may make this method no faster than that proposed by Nakamura. And like the Nakamura algorithm, for problems with three of more contacts, a numeric solution is required. The proposed method has not been extended to consider over four contacts.

2.2.4 Exact Solution Example . An exact solution to the two contact problem described in Section 2.2.2 can also be calculated using the Yoshikawa algorithm.

For the two contact problem, the matrix $\mathbf{B}_g$ is:

$$\mathbf{B}_g = \begin{bmatrix} \bar{e}_{12} \\ \bar{e}_{21} \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{2.60}$$

and the unknown grasp parameter, h_g, is a scalar. The grasp parameter is analogous to the internal force magnitude, y, of Section 2.2.2. The basis vectors for matrix $\mathbf{B}_m$ are:

$$\bar{\mathbf{e}}_{12} = \begin{Bmatrix} -1 \\ 0 \end{Bmatrix} \quad \bar{\mathbf{e}}_{21} = \begin{Bmatrix} 1 \\ 0 \end{Bmatrix} \quad \bar{\mathbf{e}}_{10} = \begin{Bmatrix} 0 \\ -1 \end{Bmatrix} \quad \bar{\mathbf{e}}_{20} = \begin{Bmatrix} 0 \\ 1 \end{Bmatrix} \tag{2.61}$$

and

$$\mathbf{B}_m = \begin{bmatrix} k_1\bar{\mathbf{e}}_{12} & \bar{\mathbf{e}}_{10} & O \\ (1-k_1)\bar{\mathbf{e}}_{21} & O & \bar{\mathbf{e}}_{20} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.62}$$

for $k_1 = -1$. The vector $\mathbf{h}_m$ may be calculated according to Equation 2.59:

$$\mathbf{h}_m = (\mathbf{W}\mathbf{B}_m)^{-1}\,\mathbf{Q} = \begin{bmatrix} 0.3333 \\ -0.5 \\ 0.5 \end{bmatrix} \tag{2.63}$$

The contact force vector, $\mathbf{F} = \mathbf{B}_m\mathbf{h}_m + \mathbf{B}_g h_g$ may now be formed,

$$\mathbf{F} = \begin{bmatrix} 0.3333 \\ 0.5 \\ 0.6666 \\ 0.5 \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \end{bmatrix} h_g \tag{2.64}$$

where the first term is analogous to the external force of Equation 2.44, though different in magnitude. An objective function and constraint equation similar to Equation 2.45 and 2.48 may be formed using the above equation for the contact forces. Solving the equivalent equations to Equation 2.49 and 2.50, the grasp parameter is calculated:

$$h_g = 0.6666 + \sqrt{(0.6666)^2 - 4(0.3333)^2(0.5)^2\left(\frac{\beta^2}{(\beta^2-1)}\right)} \tag{2.65}$$

For this particular case, $h_g = 1.7338$, and the contact forces are thus:

$$\mathbf{F} = \begin{Bmatrix} -1.4 \\ 0.5 \\ 2.4 \\ 0.5 \end{Bmatrix} \tag{2.66}$$

which are greater than those calculated using the method of Nakamura. The difference lies in the calculation of the equivalent external force. The Nakamura algorithm uses the pseudoinverse which guarantees a least squares solution of the external force, while the Yoshikawa method does not. Thus, we have illustrated for one case, the Yoshikawa solution will not result in the minimum norm of the contact forces.

2.3 Summary

We have reviewed contact force distribution methods associated with the pseudoinverse solution, proposed by Nakamura et al., and the method proposed by Yoshikawa and Nagai. In general, the method proposed by Nakamura relies on numerical solution methods which have been shown to lack real-time solution capability [63]. The Yoshikawa solution is similar, with the addition of some directional constraints, and additional computational burden.

III. Background to Fuzzy Logic

As the previous chapter pointed out, the nature of the contact force assignment problem is the solution of a generalized inverse problem. Unfortunately, these solutions are not unique [40]. The pseudoinverse solution is a convenient and relatively simple inverse to pursue. A satisfactory solution, from the point of view of real time manipulator control, of this problem with conventional mathematical tools is probably not a feasible solution in the near future [37]. As alluded to in the introductory chapter, the proposed research is grounded in artificial intelligence methods to solve this problem, specifically fuzzy logic.

3.1 Logic and Ambiguity

Binary logic was codified by Aristotle through his laws of logic based upon the concept of A and not-A; the Law of the Excluded Middle. A particle cannot be both in A and in not-A; there is no ambiguity about A. In a purist mathematical sense this is correct. However, in order to maintain this boundary of what is and what is not; artificial constraints must be applied. It is these artificial constraints which divorce mathematical rules and logic from human perception of reality. For instance, a heap of sand may be defined as a group of small particles, in close proximity, with at least x number of particles and not *one* less. Thus we have defined a heap and a non-heap; mathematically we can now differentiate between two piles of sand. One pile having x particles is a heap, and one pile having $x - 1$ particles is not a heap. The number of particles required to define a heap is only one aspect of the previous definition which is open to ambiguity.

This black and white world of sets is particularly troublesome for rule-based systems, such as knowledge-based artificial intelligence systems. Rules based on absolute dichotomy are relatively inefficient since definitions must be numerous and

overly precise. This precision leads away from generality which should be a goal for such a system [18].

A number of modifications to the bivalent logic of Aristotle have been proposed and used over the years. These modifications include the discretization of the bivalent logic. A number of different theories of multivalued logics were proposed in the early twentieth century by Bochvar, Kleene, Heyting, Reichenbach, and Lukasiewicz. Generally, these theories were initially proposed to deal with the Heisenburg uncertainty principle in quantum mechanics [29]. Lukasiewicz developed the first N-valued logic in the 1930's, where $N \geq 2$. Thus, for $N = 2$, the standard bivalent logic applies. In an N-valued logic, the truth values are assumed evenly distributed along the closed interval [0 1]. The nomenclature associated with the multivalued logics is L_N. Thus, an L_3 logic consists of truth values of { 0 1/2 1 }. In this logic, a statement could be half-true. The discretization was eventually expanded to $N = \infty$, where truth values were the real numbers in the closed interval [0 1]. However, the purpose behind the expanded logics still remained the same as the earlier bivalent logic, exact reasoning [18].

In the 1930's another logician, Bertrand Russell used the term "vagueness" to describe multivalence. In 1937, quantum philosopher Max Black published a paper on "vague" sets, which were essentially the same as the "fuzzy" sets we speak of today [29]. If the scientific community had lent more time to the concepts proposed by Black, we might have had *vague logic* 50 years ago. Lotfi Zadeh first proposed the basic theory of fuzzy sets in his 1965 paper 'Fuzzy Sets,' in which he referred to the work of Kleene. The paper on fuzzy sets later served as the basis for the fuzzy logic now common today. The math of fuzzy sets used the same algebra worked out by Lukasiewicz in the 1930's, though today it has expanded.

The essential premise of fuzzy sets is the contradiction of Aristotle's Law of Excluded Middle. A particle may exhibit a level of *membership* in one or more sets. Thus a half eaten apple has a level of membership in the set of apples and in the

sets of non-apples. Depending on the characteristics of the domain of interest, the summation of the degree to which the half-eaten apple belongs to other sets may exceed unity. This is a difference between fuzzy sets and other exact multivalence set theories.

3.2 Fuzzy Sets and Fuzzy Logic

Zadeh showed how the concepts of fuzzy sets and fuzzy logic mimic the linguistic constructs of human language and how such concepts may allow reasoning akin to humans [28]. Rules based on ambiguous linguistic concepts may be used with fuzzy logic; where fuzzy logic provides the numeric mechanism by which degrees of truth of premises (antecedents) are manipulated through the rules to provide conclusions (consequents).

The definition of membership is the *degree* of the truth of the statement: x is a member of fuzzy set $\mathcal{A}$. The membership value is in the closed interval [0 1], 0 indicating complete falsehood and 1 indicating complete truth. The finite fuzzy set $\mathcal{A}$ can be represented as the set of ordered pairs $\mathcal{A} = (x, \eta_A(x)) \,|x \in X$, where X is the domain and $\eta_A(x)$ is the membership function. Figure 3.1 demonstrates the difference between fuzzy and binary (crisp) sets by plotting the membership functions for the sets of medium and tall people. There are many subjective definitions of the makeup of a fuzzy set which comprise the domain of interest. In Figure 3.1 sets of both medium and tall people are described over the domain of height. The *support* of each fuzzy set constitutes the portion of the domain where the membership values are greater than zero. A common approach is the use of triangular membership functions as this helps to speed up the numeric process [58].

Fuzzy logic is concerned with *approximate reasoning* rather than exact multivalued logic as proposed by Lukasiewicz et al. Fuzzy reasoning consists of the inference of a possibly imprecise conclusion from a set of possibly imprecise premises. This is the action involved with most human reasoning. There are basically two approaches

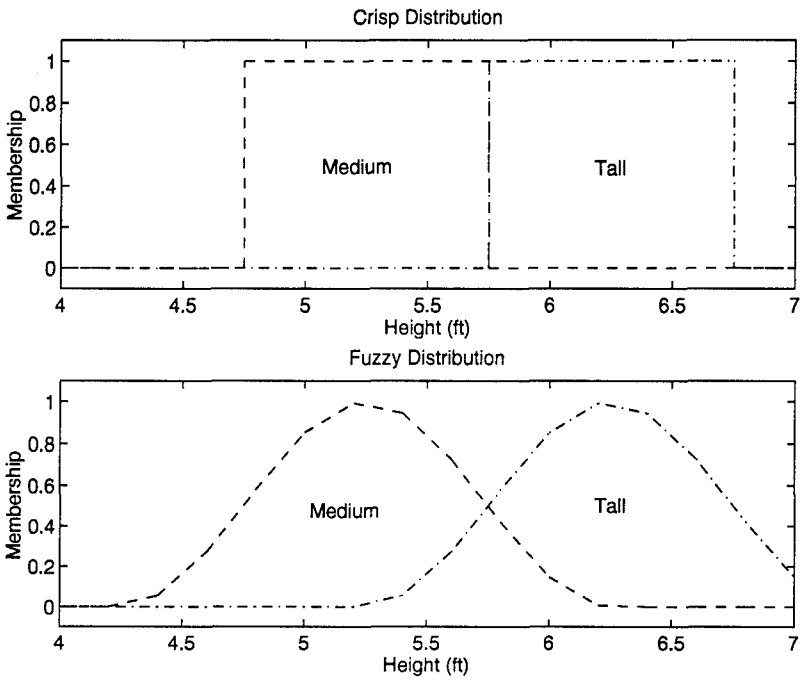

Figure 3.1 Conventional and fuzzy sets

to using fuzzy logic to solve input-output mapping problems: develop a logical rule base similar to a rule base a human might use to solve the same problem, or use acquired data of correct input-output pairs to develop the rule base by learning the mapping. The learning method strictly uses the current data to develop a fuzzy logic system capable of generating the same output as the original system [55, 28].

This research will use a heuristic approach to the fuzzy logic rules and domains. If somewhat promising, future research may strive for adaptive or learning systems which seek to minimize some objective function. The primary concern of this research is the development of tools for quick numeric solutions and will not strive to explicitly produce a mathematically optimum solution of some objective function. The rules and the domains of the relevant parameters will be derived in the spirit of traditional expert systems.

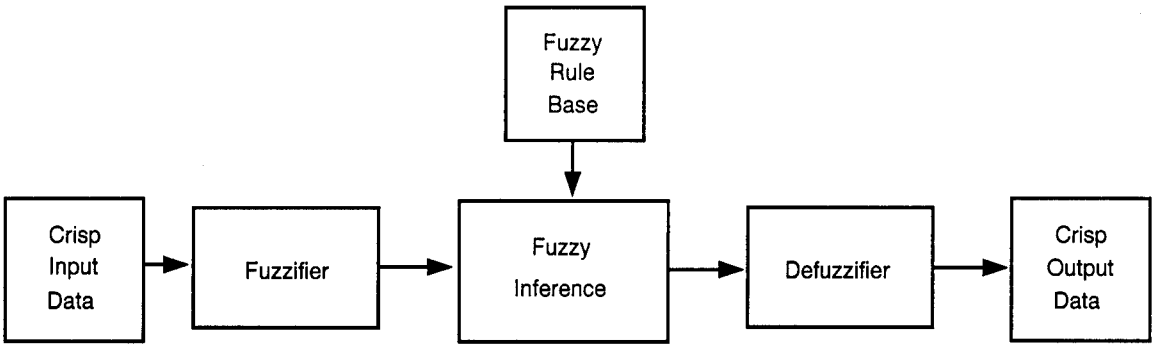

Figure 3.2 Fuzzy logic system

3.3 Fuzzy Logic Mechanics

A basic fuzzy logic system may be described by Figure 3.2. The inputs are generally discrete values obtained from digital inputs or analog inputs, the latter obtained from A/D devices. Note: Some fuzzy systems have been developed to take advantage of direct analog inputs. The inputs are then categorized as fitting each of the input membership functions to some degree. This process is commonly referred to as *fuzzification* [64, 11]. The membership functions which span the input domain are defined and fixed *a priori* for non-adaptive systems. The fuzzy inputs are then fed across the fuzzy rule-base, where an inference is made from each instance of the data. The inference mechanism is based on the rules which form the rule-base.

The rules consist of fuzzy antecedents, aggregation operators, and fuzzy consequents. The antecedents and consequents are the fuzzy input and output sets. The aggregation operators are from the *t-norm* and *t-conorm* set of operators which generalize the intersection (conjunction) and union (disjunction) operations respectively [57]. An example of a rule is:

$$IF\ A\ is\ U_{Ai}\ AND\ B\ is\ U_{Bi}\ THEN\ C\ is\ U_{Ci} \tag{3.1}$$

where A is the variable of the domain U_A, B is the variable of the domain U_B, and C is the variable of the domain U_C. U_{Ai}, U_{Bi}, and U_{Ci} denote specific member-

ship functions (fuzzy variables) of their respective domains. Equation 3.1 uses the conjunctive aggregation operator, AND. Similarly, OR is a disjunctive operator. The mechanism by which fuzzy sets are aggregated is not unique. However, all such mechanisms do fall within the definitions of *t-norm* and *t-conorm* defined respectively as the mappings $T : [0,1] \times [0,1] \rightarrow [0,1]$ and $S : [0,1] \times [0,1] \rightarrow [0,1]$ such that these mappings meet the following criteria [28]:

$$\begin{array}{ccc} T(a,b) = T(b,a) & S(a,b) = S(b,a) & \text{Commutativity} \\ T(a,T(b,c)) = T(T(a,b),c) & S(a,S(b,c)) = S(S(a,b),c) & \text{Associativity} \\ T(a,b) \geq T(c,d) & S(a,b) \geq S(c,d) \; if \; a \geq c \; and \; b \geq d & \text{Monotonicity} \\ T(a,1) = a & S(a,0) = a & \text{Identity} \end{array} \tag{3.2}$$

where a and b are the degrees of membership in their appropriate membership functions. The Min/Max operators,

$$Max(a,b) = \frac{a+b+|a-b|}{2}$$
$$Min(a,b) = \frac{a+b-|a-b|}{2}$$

are commonly used t-norm/t-conorm mappings [57].

Though many other operators exist, Min/Max will be used for this research. Thus, the conjunctive operator is:

$$Min(\eta_A(x_A), \eta_B(x_B)) \tag{3.3}$$

where x_A and x_B are the input values associated with the A domain and B domains, respectively; determines the weight of the associated fuzzy consequent. Once the fuzzy consequent and the weight associated with it are determined for all fuzzy input pairs, one must derive a non-fuzzy output. This process is known as defuzzification and again, there are many methods used for this purpose [58, 57, 13]. This research

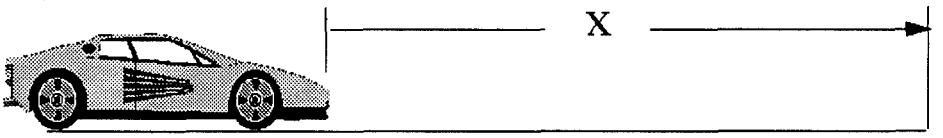

Figure 3.3 One dimensional example

will use the weighted average of fuzzy singletons which has a significant reduction in computational costs compared with other common methods [51]. A summary of the design parameters which must be defined include: parameters which constitute the inputs, fuzzy domains which span the input and output space, fuzzy rule-base, aggregation operators, and defuzzification procedure. A simple example of this process follows.

3.4 Example of Fuzzy Control

A fuzzy logic system consisting of heuristically derived rules, consists of a rule base, which is an n^{th} order system of relationships among the input parameters and output parameters. n being the number of input parameters in the rule base. Typically, the rule base may be envisioned as an n^{th} order matrix of cells. Each cell corresponds to a fuzzy output set given the input fuzzy arguments. The following example models simplistic position control using acceleration inputs for an automobile.

This example will concern the single dimensional problem controlling an automobile to a given position through commanded changes in acceleration. Assume the input variables for an automobile controller can be reduced to two inputs, distance to the target, x and current velocity $v = \dot{x}$. Figure 3.3 illustrates the problem. Let the output be the acceleration $a = \ddot{x}$. Assume the domains of interest are:

$$x : \begin{bmatrix} -10 & 10 \end{bmatrix}$$

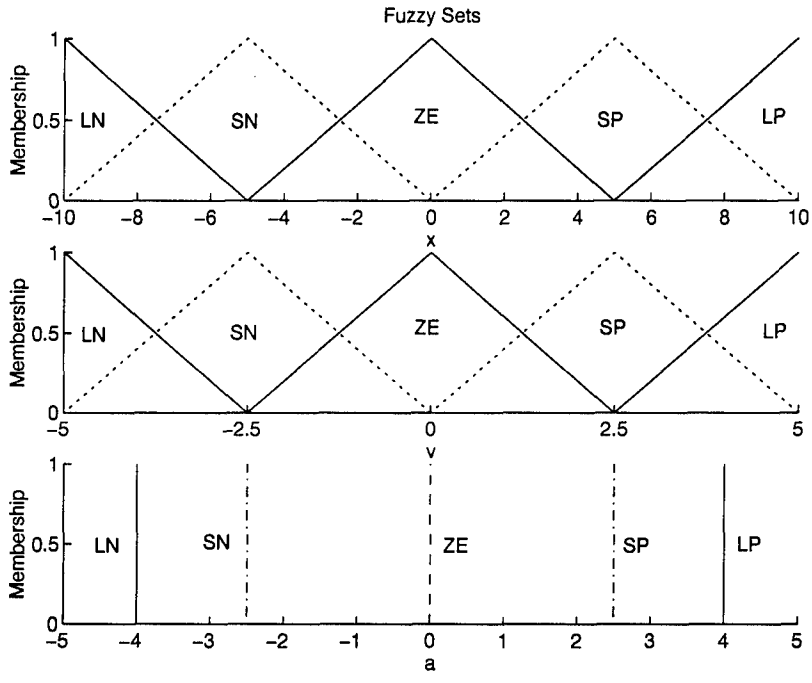

Figure 3.4 Example fuzzy antecedent and consequent sets

$$v : \begin{bmatrix} -5 & 5 \end{bmatrix}$$
$$a : \begin{bmatrix} -5 & 5 \end{bmatrix}$$

Further, assume the domains are divided among the five fuzzy sets: large negative (LN), small negative (SN), zero (ZE), small positive (SP), and large positive (LP). These domains are illustrated in Figure 3.4. Using the above assumptions, the structure of the rule-base is defined. The rule-base will consist of a 25 element matrix, which constitutes all the possible combinations of fuzzy set inputs. The acceleration domain is made up of fuzzy singletons [51]. The output membership functions are unit impulses, which when coupled with a weighted average inference process, is commonly referred to as the Sugeno-style fuzzy inference. The heuristic rules used to determine the output for each cell derive from various permutations of the rule below:

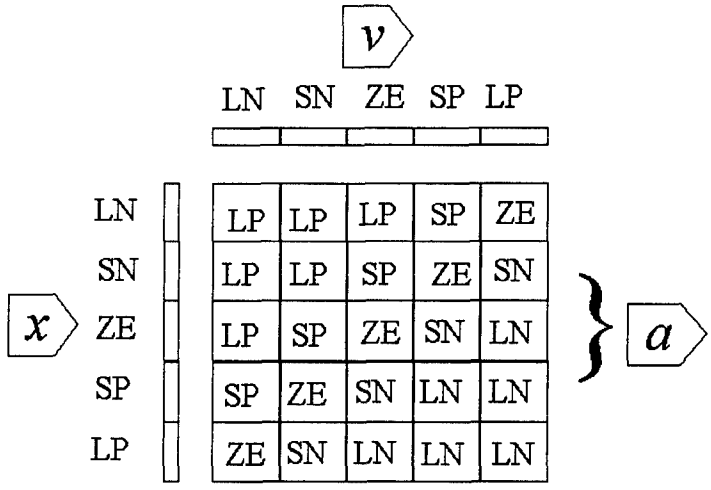

Figure 3.5 Example rulebase

$$IF\ v\ is\ LP\ AND\ x\ is\ SP\ THEN\ a\ is\ LN \tag{3.4}$$

The control surface of these heuristically derived rules, is illustrated in Figure 3.5.

The operation of the inference mechanism, where an instance of inputs is mapped to the output range through the rule-base, is described next. Given an input pair $v = -2.0,\ x = 3.3$, the first step in the inference process is the fuzzification of the inputs. The velocity and distance membership values are: $\{0.0, 0.8, 0.2, 0.0, 0.0\}$ and $\{0.0, 0.0, 0.34, 0.66, 0.0\}$ respectively. Or, $\eta_v^{SN}(-2.0) = 0.8$, $\eta_v^{ZE}(-2.0) = 0.2$, $\eta_x^{ZE}(3.3) = 0.34$, $\eta_x^{SP}(3.3) = 0.66$, with the membership in the other functions zero. For each cell in the rule-base matrix, the weight of the output cell is determined by the value, $Min(\eta_v(v), \eta_x(x))$. Thus, many of the cells result in zero output for a given input set $\{\ x\ \ v\ \}$. Figure 3.6 illustrates the weight associated with each of the input membership functions. In general, for two input systems, only four rules of the rulebase will have greater than zero output.

$$\eta(SN, ZE) = Min(0.8, 0.34) = 0.34$$
$$\eta(SN, SP) = Min(0.8, 0.66) = 0.66$$

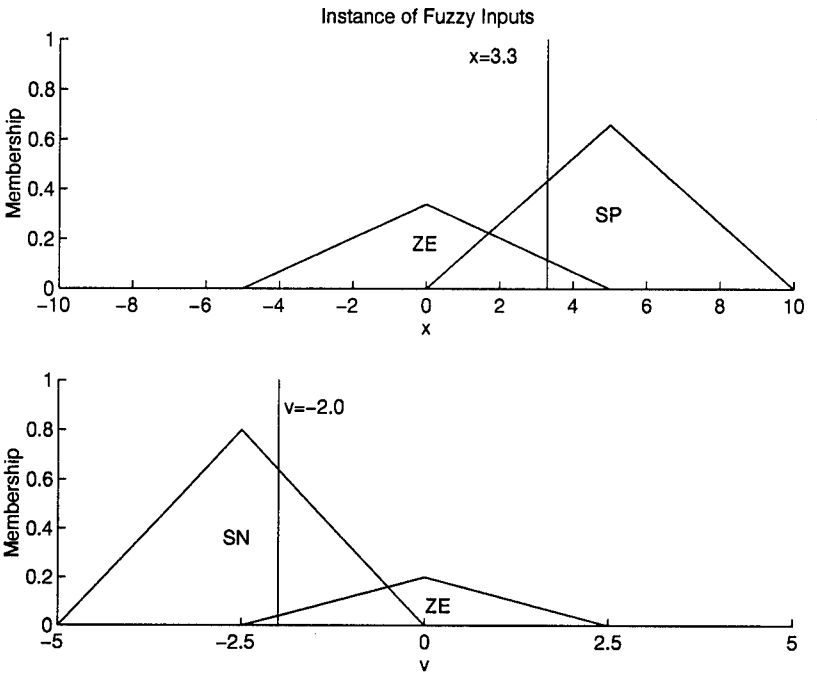

Figure 3.6 Instance of $x = 3.3$, and $v = -2.0$

$$\eta(ZE, ZE) = Min(0.2, 0.34) = 0.2$$

$$\eta(ZE, SP) = Min(0.2, 0.66) = 0.2$$

The final output control value is the weighted average of the four consequents.

$$y^* = \frac{\sum_{i=1}^{m} y_i \eta_{ai}}{\sum_{i=1}^{m} \eta_{ai}} \tag{3.5}$$

where m is the number of fuzzy consequents, y_i is the i^{th} fuzzy singleton value, η_{ai} is the weight of the i^{th} consequent

The SIMULINK model which was used to simulate this controller is illustrated in Figure 3.7 [1, 2]. Two simulations are illustrated for two different initial conditions in Figures 3.8 and 3.9.

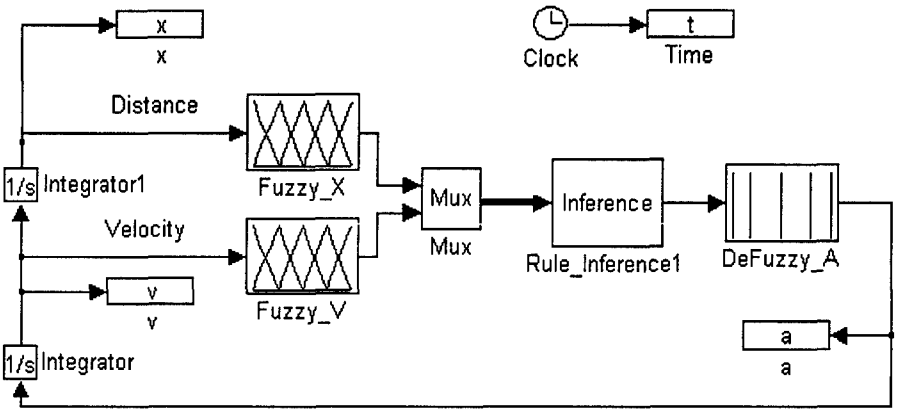

Figure 3.7 Simulink model of 1-D example

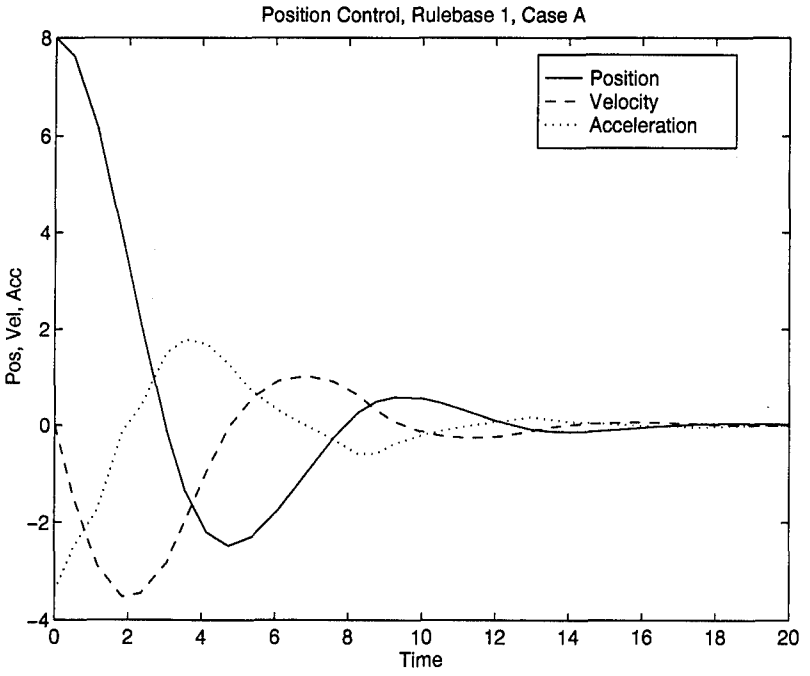

Figure 3.8 Position control results, Case A

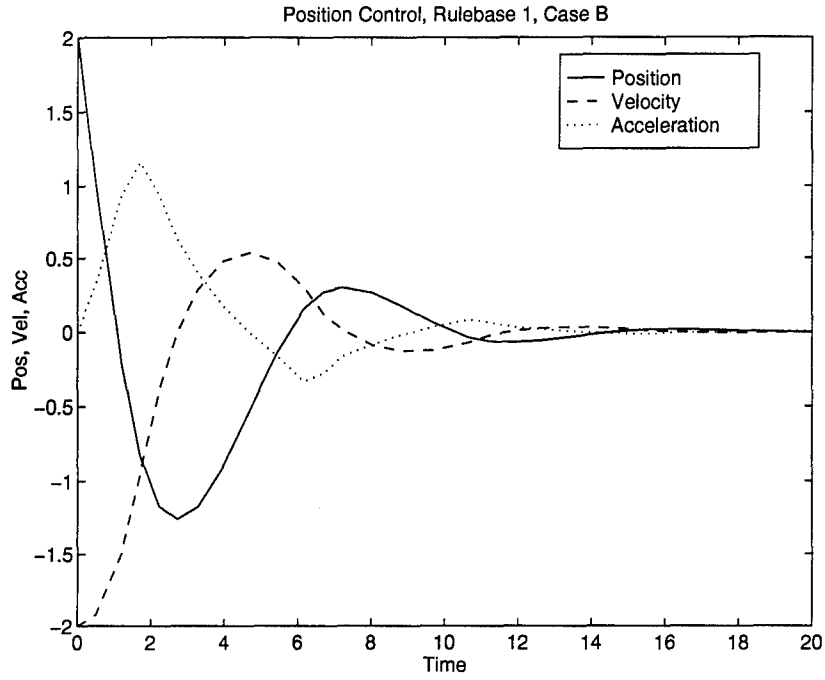

Figure 3.9 Position control results, Case B

The performance characteristics of the example position controller can be easily changed by changing the values of the cells in the rule-base. The modified rule-base along with the change in controller performance are illustrated in Figures 3.10, 3.11, and 3.12

Note, this particular fuzzy control model functions similarly to a conventional PD controller. Simple fuzzy control models using inputs and outputs equivalent to errors, change in errors, and sum of errors approximate various forms of PI, PD, and PID control schemes [45, 48, 57].

3.5 Summary

In this chapter, a brief history of the concept of fuzzy sets and logic have been presented. Also presented were the mechanics of a simple fuzzy logic controller, including concepts surrounding construction of the system. The basic elements in the fuzzy controller include definition of the input parameters in terms of a fuzzy domain,

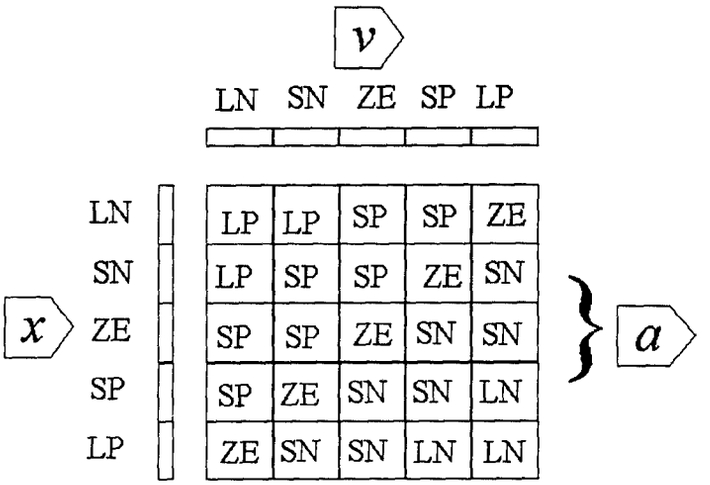

Figure 3.10 Modified rulebase

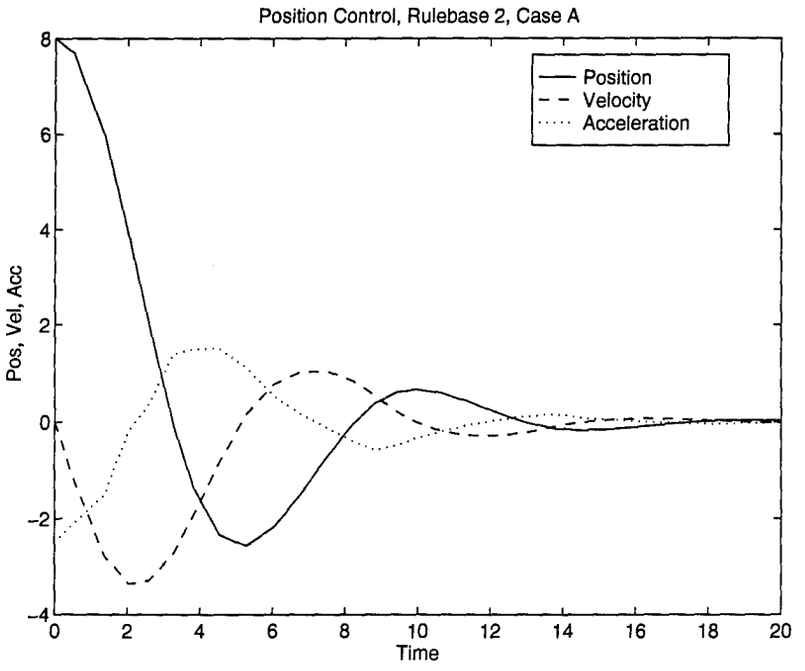

Figure 3.11 Modified position control results, Case A

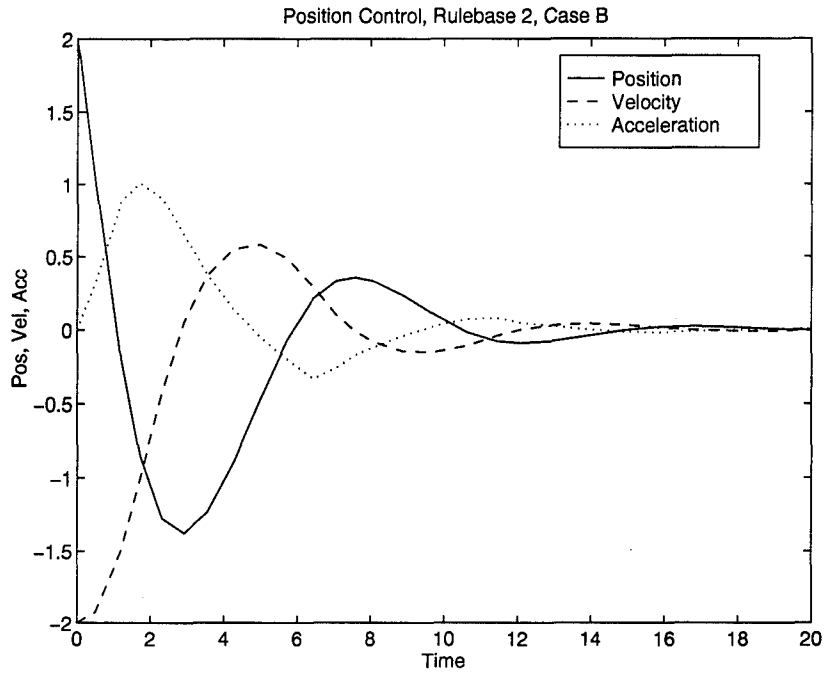

Figure 3.12 Modified position control results, Case B

the rule-base by which the inference mechanism can operate, and the determination of the final consequent response. These elements are common to most fuzzy control systems. The decisions concerning the details of the shape of the domains, the aggregation operators, the rule-base, and the consequent analysis are flexible to the extent that they can be compared with tuning parameters of conventional control architectures.

IV. Fuzzy Logic Reactive System

This research concerns the development and validation of an unconventional approach to the problem of contact force assignment. Conventional methods based on global optimization attempt to satisfy the stated objective functions while simultaneously considering all contact forces. These methods require numeric optimization solutions for all but very simple contact configurations. This research will focus on an alternative method which evaluates each internal force pair in turn and, through fuzzy logic methods, then makes a global iterative adjustment of the internal forces.

4.1 FLRS Concept

The FLRS algorithm is an iterative one. Starting with the known external force solution, declare an error to exist among the contact forces which do not satisfy their local frictional constraints. Next, evaluate how well each internal force can reduce the associated contacts errors and proportionally weight each internal force accordingly. All the internal forces are then scaled such that, when added to the previous contact forces, they minimize the contact error for at least one contact. The contact used to determine the scale of the internal force increment is the contact whose error is most readily reduced by the application of internal forces. The iterative process is continued until *all contacts satisfy the friction constraints.* Since only internal forces are being added to the external force solution, the object wrench due to the contact forces remains the same, i.e. the addition of null space forces to the pseudoinverse solution cannot change the object wrench.

4.2 FLRS Parameters

Figure 4.1 illustrates how the FLRS algorithm calculates the internal forces for a given grasp configuration and object wrench. No knowledge of a previous solution is assumed. Before entering into the iterative portion of the algorithm, the external

forces are calculated using the pseudoinverse of the grasp matrix. The first element in the iterative portion of FLRS is the establishment of the contact errors which are to be resolved, as indicated in Figure 4.2. The i^{th} contact force error for the t^{th} iteration, $\mathbf{er\_\hat{f}}_i(t)$, is the vector from the current contact force, $\hat{\mathbf{f}}_i(t)$, to the closest point on the friction cone, in the i^{th} coordinate frame,

$$\mathbf{er\_\hat{f}}_i(t) = \begin{bmatrix} dx_i(t) \\ dx_i(t) \cdot \mu \cdot \cos(\phi_i(t)) \\ dx_i(t) \cdot \mu \cdot \sin(\phi_i(t)) \end{bmatrix} - \begin{Bmatrix} \hat{f}_{ix}(t) \\ \hat{f}_{iy}(t) \\ \hat{f}_{iz}(t) \end{Bmatrix} \tag{4.1}$$

where

$$dx_i(t) = \max\left(\frac{\hat{f}_{ix}(t) + \mu(\hat{f}_{iy}(t)\cos(\phi_i(t)) + \hat{f}_{iz}(t)\sin(\phi_i(t)))}{1+\mu^2}, 0 \right) \tag{4.2}$$

and

$$\phi_i(t) = a\tan 2(\hat{f}_{iz}(t), \hat{f}_{iy}(t)) \tag{4.3}$$

The superscript '$\hat{}$' indicates force variables which exist in the iterative portion of FLRS only. The x-coordinate, in the i^{th} frame for the t^{th} iteration, of the closest point on the friction cone to the current value of contact force, is denoted as $dx_i(t)$. The max operation of Equation 4.2 is necessary to ensure $dx_i(t) \geq 0$ even for $\hat{f}_{ix}(t) \ll 0$.

Given an initial external contact force, $\mathbf{f}ext_i$, and contact error, $\mathbf{er\_\hat{f}}_i(1)$, if internal forces were added to eliminate the contact error as defined, and ignoring other contacts, the additional internal force would result in the minimum norm contact force. The least internal force required to satisfy the friction constraints for the given contact, is $\mathbf{er\_\hat{f}}_i(1)$. However, other contact errors exist and their resolution may conflict in terms of the internal forces necessary to resolve the contact errors. A method of quantifying the ability of a given internal force to constructively alter the error in a given contact force, $\mathbf{er\_\hat{f}}_i(t)$, is needed. Figure 4.3 illustrates the method. The internal force unit vector, from contact i to contact j is denoted $\mathbf{e\_I}_{ij}$ and is

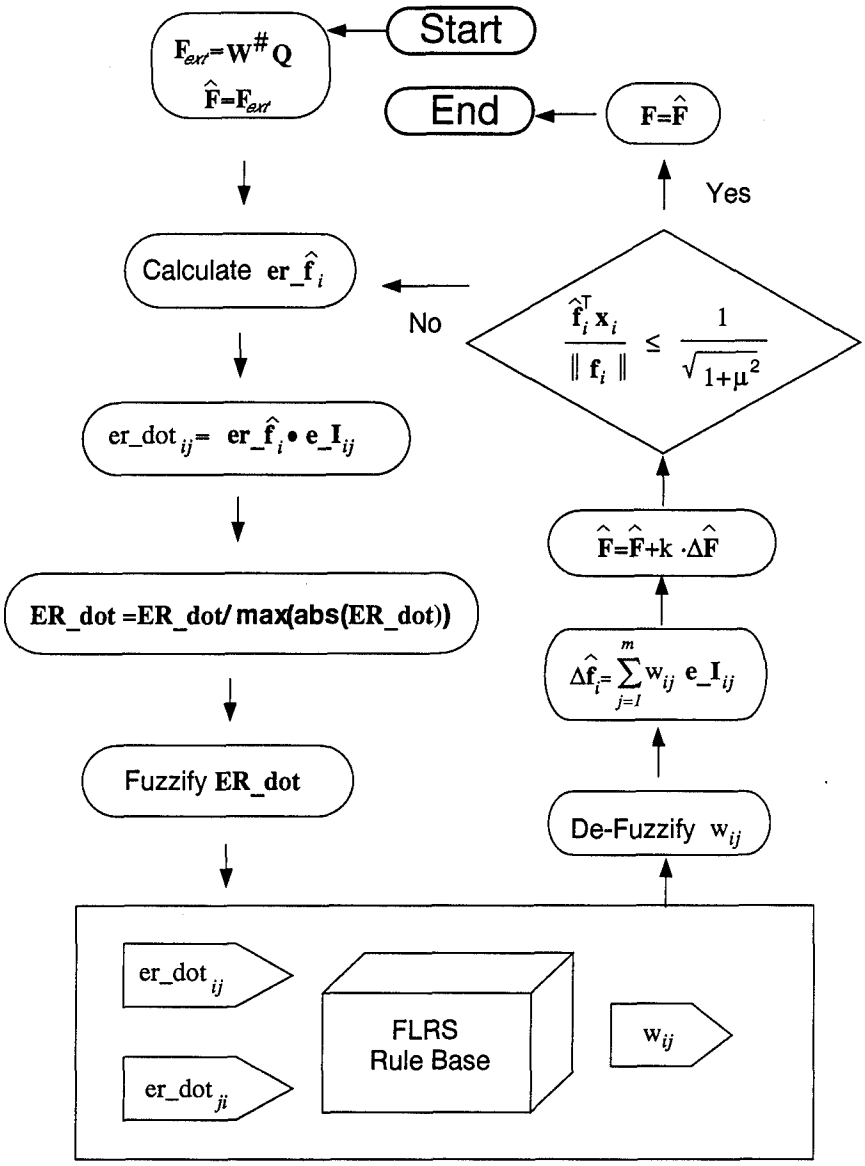

Figure 4.1 FLRS internal force algorithm

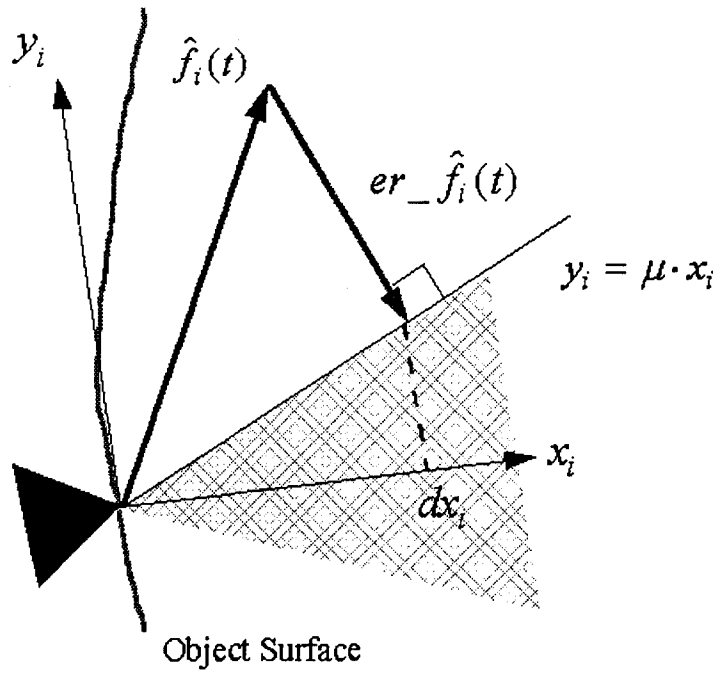

Figure 4.2 Definition of $\mathbf{er\_\hat{f}_i(t)}$

of equal magnitude and opposite direction to $\mathbf{e\_I}_{ji}$. The vector dot product of an internal force unit vector with an associated error in contact force, $\mathbf{e\_I}_{ij} \bullet \mathbf{er\_\hat{f}}_i(t)$ and $\mathbf{e\_I}_{ji} \bullet \mathbf{er\_\hat{f}}_j(t)$, are denoted by er_dot_{ij} and er_dot_{ji} respectively. The symbol '$\bullet$' denotes vector dot product. It is to be understood that er_dot_{ij} is a function of the iteration, t, of the algorithm.

The er_dot_{ij} scalar quantifies how effectively a particular internal force can resolve an associated contact force error. Initially, $er\_dot_{ij} \in (\ -\infty \quad \infty\ )$; however, the FLRS algorithm will normalize this domain, during every iteration, such that $er\_dot_{ij} \in (\ -1 \quad 1\ )$ by letting $\mathbf{ER\_dot} = \mathbf{ER\_dot} / \max(abs(\mathbf{ER\_dot}))$, where $\mathbf{ER\_dot} \in \Re^{2N_I}$ denotes the vector of all er_dot_{ij}. Recall, N_I is the number of internal force pairs, $(m^2 - m)/2$.

So far we've addressed only the input, or antecedent, parameters which feed into the fuzzy inference portion of FLRS. The goal of FLRS is to determine a solution to the frictional contact problem by iteratively adding necessary increments of

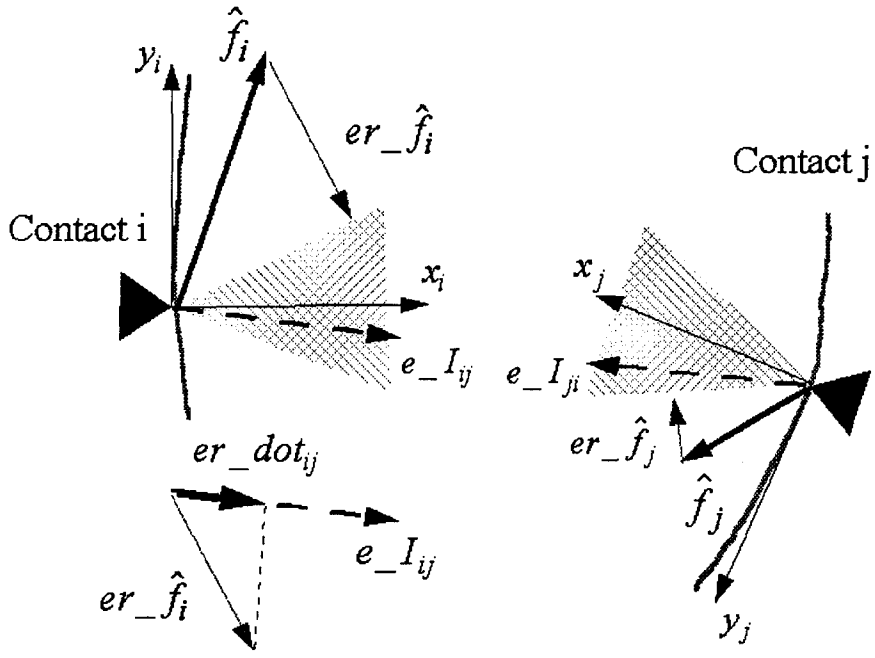

Figure 4.3 Definition of er_dot_{ij}

internal force. Thus, the output, or consequent, parameter will be the relative weight associated with a given internal force, $w_{ij}(t)$. Fuzzification of the antecedents and defuzzification of the consequents, using the method of Sugeno, will be accomplished as described in Chapter III.

4.3 FLRS rulebase

The FLRS fuzzy inference portion operates on each internal force pair, in turn. A relative evaluation as to the merit of an increase in one internal force pair over another must be made. This will be accomplished through fuzzy inference using a heuristically formed rulebase. The basis for the rulebase in evaluating the merit of each internal force is straightforward and two-dimensional. If the effects of all internal forces were considered simultaneously, the dimension of the rulebase would increase to $2N_I$ and would dramatically increase the computations necessary for a solution as well as the complexity of the rulebase. Assuming the FLRS algorithm

converges, a two-dimensional rulebase will provide an improvement in solution speed compared with the $2N_I$-dimensional rulebase.

Given the i^{th} internal force and the associated er_dot_{ij} and er_dot_{ji}, the outline of the rulebase is:

$$If\ er\_dot_{ij}\ is\ PositiveLarge\ and\ er\_dot_{ji}\ is\ PositiveLarge\ then\ w_{ij}(t)\ is\ Large \tag{4.4}$$

Equation 4.4 illustrates the concept that if er_dot_{ij} and er_dot_{ji} are relatively large positive numbers, then an increase in the internal force associated with both i and j contacts will greatly decrease the contact force error for both contacts. Likewise, if both er_dot_{ij} and er_dot_{ji} are relatively large negative numbers, then a decrease in the internal force associated with both i and j contacts will greatly decrease the contact force error for both contacts. Also, the relative magnitudes of the input er_dot will dictate the relative magnitude of the output, $w(t)$. For instance, if both er_dot_{ij} and er_dot_{ji} are relatively small positive numbers, then an increase in the internal force associated with both i and j contacts will slightly decrease the contact force error for both contacts.

Figure 4.4 illustrates the FLRS rulebase. The rulebase is symmetric with respect to the ij and ji inputs. The diagonal elements, elements for which the ij and ji inputs are equal, represent the qualitative arguments presented above. The off diagonal terms represent the heuristic notion of tradeoffs. For example, if er_dot_{ij} is a large positive number, reflecting the fact that an increase in the internal force will greatly decrease the i^{th} contact error, and er_dot_{ji} is a relatively small negative number, then a moderate increase in the internal force associated with both i and j is warranted as the benefits of such an increase outweigh the detriments. Again, there is a full range of relative values for both the i^{th} and j^{th} contacts. At this point some decisions, concerning the structure of the fuzzy inference portion of FLRS, need to be made.

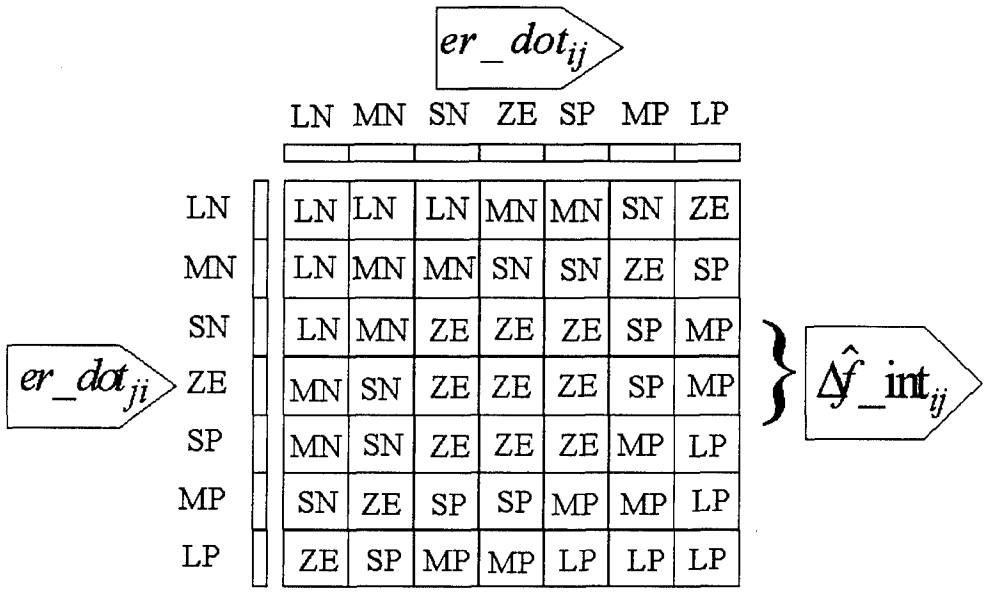

Figure 4.4 FLRS rulebase

As Figure 4.4 illustrates, the number of input and output fuzzy sets has been defined to be seven. The number of fuzzy sets reflects, to some degree, the level of resolution required to adequately express the heuristic rules described above. Figure 4.5 illustrates the antecedent and consequent domain and their associated fuzzy sets. These sets are symmetric about zero. The support of the antecedent sets, the interval of the domain where the membership values are greater than zero, is graduated with the support widening for the sets closer to the domain extremes. This allows finer resolution of the antecedents when they are relatively small. The output domain is defined for seven fuzzy singleton sets, as per the Sugeno method of defuzzification [51]. Once the limits on the fuzzy domains were established, the shape and mean of the fuzzy sets were chosen to resemble examples of fuzzy inference systems common in the literature, such as Ross [52]. The Sugeno method of inference was chosen to speed the calculation of the fuzzy inference consequent.

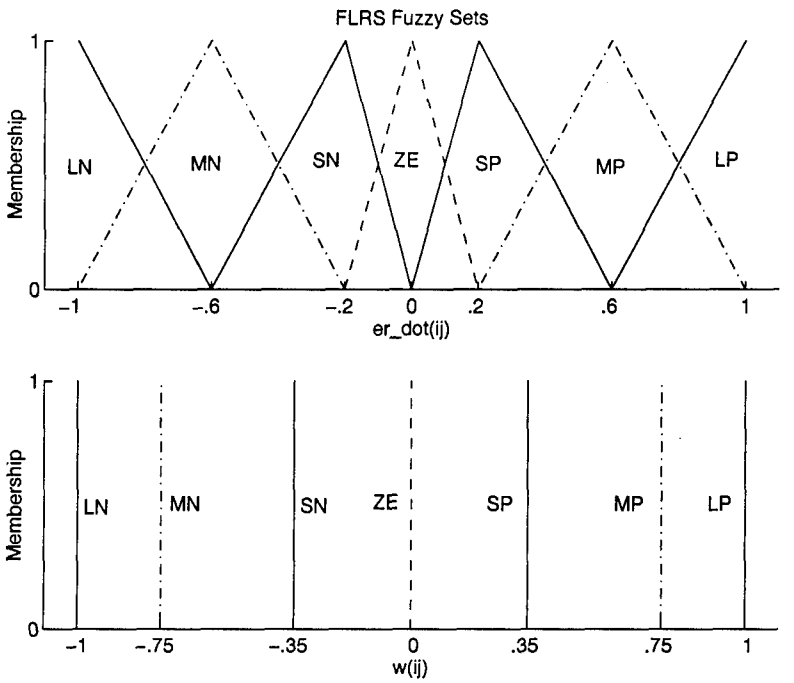

Figure 4.5 FLRS fuzzy antecedent and consequent sets

4.4 Change in Internal Force

After each internal force has received a weight, $w_{ij}(t)$, the change to the internal force associated with the i^{th} contact is calculated as:

$$\boldsymbol{\Delta}\hat{\mathbf{f}}_i(t) = \sum_{j=1, j\neq i}^{m} w_{ij}(t) \cdot \mathbf{e\_I}_{ij} \tag{4.5}$$

where '$\cdot$' signifies scalar product. The change in internal forces are next scaled and added to the current contact forces to form the updated contact forces, $\hat{\mathbf{F}}(t)$.

$$\hat{\mathbf{F}}(t) = \hat{\mathbf{F}}(t) + k(t) \cdot \boldsymbol{\Delta}\hat{\mathbf{F}}(t) \tag{4.6}$$

where

$$k(t) = \left(\boldsymbol{\Delta}\hat{\mathbf{f}}_L(t)\right)^{\#} \mathbf{er\_}\hat{\mathbf{f}}_L(t) \tag{4.7}$$

and L is the contact with the largest value of $\Delta\hat{\mathbf{f}}(t)_i \bullet \mathbf{er\_}\hat{\mathbf{f}}_i(t)$. The symbol '#' denotes the pseudoinverse operator. This scale factor has the effect of reducing $\mathbf{er\_}\hat{\mathbf{f}}_L(t)$ in the next iteration and reducing the $\mathbf{er\_}\hat{\mathbf{f}}_i(t)$ of the other contacts by lesser amounts. This is discussed in more detail in Section 5.1.2. The above definition of the scale factor, $k(t)$ promotes the *orderly* solution of each contact force error. *Orderly*, in this case, means that the contacts which will reach the friction constraint surface first, due to the change in internal force, will control the solution process. As each contact force, $\hat{\mathbf{f}}_i(t)$ is pushed into the friction cone by the addition of internal forces, that contact ceases to have a contact force error. However, during following iterative cycles, if necessary, the incremental addition of internal force may cause a previously zero contact force error to become non-zero. The FLRS iteratively cycles until all contact forces lie inside their respective friction cones. Bounds on $k(t)$, for solution convergence, will be established in Chapter V.

The scale factor $k(t)$ may be viewed as a gain, dependent on the change in the L^{th} contact internal forces and error in force. This algorithm is analogous to a disturbance rejection proportional controller which seeks to reduce an error to zero. Accordingly, this control scheme may take an excessive number of iterative steps to reach a zero error state. Another source of slowed convergence can result when $k(t)$ is repeatedly established by the same contact. The issue of convergence speed enhancements to the FLRS algorithm, will be addressed in Chapter VI.

4.5 Summary

The FLRS algorithm iteratively weights the use of internal forces without complete knowledge of all the contact error states. This is the fundamental difference between the FLRS and other optimal forms of grasp force assignment which consider all error states simultaneously. Both the FLRS and other grasp force assignment methods begin with some form of external force solution, upon which internal forces

are added until the contact constraints are met. Useful extensions to the basic FLRS algorithm are presented in Appendix A.

V. FLRS Convergence Characterization

The grasp force assignment algorithm described in Chapter IV must be validated. This chapter will characterize the limitations on the parameters governing the FLRS solution such that solution convergence is assured for both the two contact case and multicontact ($m > 2$) case. Observations concerning the characteristics of the FLRS solution with respect to the Nakamura optimal solution will also be made.

5.1 Convergence Analysis

The analysis which follows will be planar though it should be extendible to spatial cases. The objective of this analysis is to characterize the functions and parameters used in the FLRS algorithm which will ensure solution convergence. The convergence criteria which will be satisfied is the decrease, at each iterative step, of the sum of the norms of the error in contact force over all contacts,

$$\sum_{i=1}^{m} \left\| \mathbf{er\_f}_i(t+1) \right\| < \sum_{i=1}^{m} \left\| \mathbf{er\_f}_i(t) \right\| \tag{5.1}$$

where t is an integer and indicates the iteration number of the FLRS algorithm. It is well known that a decreasing and bounded sequence is convergent [5]. As described in Chapter IV the FLRS solution is iterative in nature and starts with the external force solution,

$$\hat{\mathbf{F}}(1) = \mathbf{F}ext = \mathbf{W}^{\#}\mathbf{Q} \tag{5.2}$$

where $\mathbf{W}$ is the grasp matrix associated with the given grasp configuration and $\mathbf{Q}$ is the commanded object wrench..

This analysis will assume that the grasp configuration is consistent with concepts regarding force closure, [62]. Without a force closure grasp, there will exist some commanded object wrench for which the FLRS algorithm will not converge to a solution, nor would any other algorithm.

This analysis will proceed by first establishing the contact force relationships from one iteration to the next. Second, contact force errors will be defined. Third, the square of the Euclidean norm of these errors will be developed for the t^{th} and $(t+1)^{th}$ solution iterations, as in Equation 5.1. And finally, the requirements on the contact geometry and solution parameters will be derived based on conformance with solution convergence. Note, this convergence analysis uses subscript notation which should not be confused with standard forms of tensor notation.

Given a contact force vector, $\widehat{\mathbf{F}}(t)$ at iteration t, the next contact force solution will be:

$$\widehat{\mathbf{F}}(t+1) = \widehat{\mathbf{F}}(t) + k(t)\Delta\widehat{\mathbf{F}}(t) \tag{5.3}$$

where

$$\widehat{\mathbf{F}}(t) = \left[\begin{array}{cccc} \widehat{\mathbf{f}}_1^T(t) & \widehat{\mathbf{f}}_2^T(t) & \ldots & \widehat{\mathbf{f}}_m^T(t) \end{array} \right]^T \tag{5.4}$$

$$\Delta\widehat{\mathbf{F}}(t) = \left[\begin{array}{cccc} \Delta\widehat{\mathbf{f}}_1^T(t) & \Delta\widehat{\mathbf{f}}_2^T(t) & \ldots & \Delta\widehat{\mathbf{f}}_m^T(t) \end{array} \right]^T \tag{5.5}$$

and where $\Delta\widehat{\mathbf{F}}(t)$ consists of internal force components only and $k(t)$ is an overall scale factor to be defined. A diagram of a three contact force assignment problem and relevant nomenclature is shown in Figure 5.1. The contact coordinate systems are such that the inward pointing normal is coincident with the local x-axis, the z-axis is perpendicular and out of the page, and the y-axis is consistent with a right-handed coordinate system.

The solution for the $(t+1)^{th}$ iteration in contact force for the i^{th} contact is:

$$\hat{\mathbf{f}}_i(t+1) = \hat{\mathbf{f}}_i(t) + k(t) \cdot \Delta\hat{\mathbf{f}}_i(t) \tag{5.6}$$

where

$$\Delta\hat{\mathbf{f}}_i(t) = \sum_{j=1}^{m} w_{ij}(t) \cdot \mathbf{e\_I}_{ij} \tag{5.7}$$

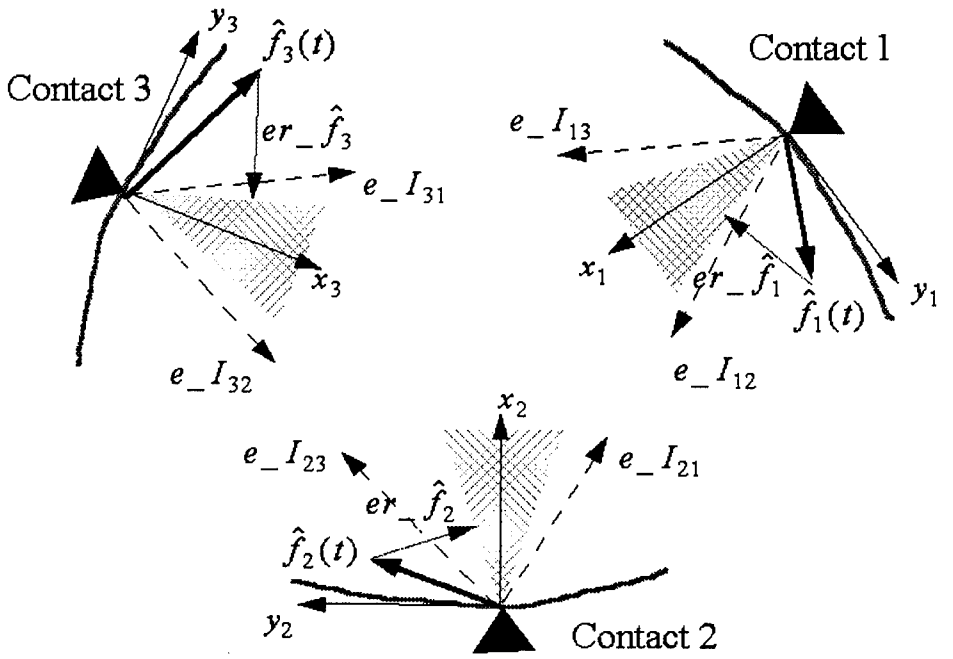

Figure 5.1 FLRS solution nomenclature for three contact problem

and $w_{ij}(t)$ is a weight associated with the internal force between the i^{th} and j^{th} contacts. The term $\mathbf{e\_I}_{ij}$ is the unit vector from contact i to contact j, in the i^{th} coordinate frame. It is to be understood that the summation does not include $j = i$ as there is no $w_{ii}(t)$. Let

$$w_{ij}(t) = w_{ij}(er\_dot_{ij}(t) + er\_dot_{ji}(t)) = w_{ji}(er\_dot_{ij}(t) + er\_dot_{ji}(t)) \tag{5.8}$$

where $w_{ij}(t) \in [\ -1 \quad 1\ ]$ is an increasing function of $er\_dot_{ij}(t)$ and $er\_dot_{ji}(t)$ which passes through the origin, i.e. $x \geq y \rightarrow w_{ij}(x) \geq w_{ij}(y)$. Also, $er\_dot_{ij}(t) \in [\ -1 \quad 1\ ]$, where

$$er\_dot_{ij}(t) = \left(\mathbf{e\_I}_{ij} \bullet \mathbf{er\_\hat{f}}_i(t)\right) / C_{dot} \tag{5.9}$$

and

$$C_{dot} = \max_{ij}(\mathbf{e\_I}_{ij} \bullet \mathbf{er\_\hat{f}}_i(t)) \tag{5.10}$$

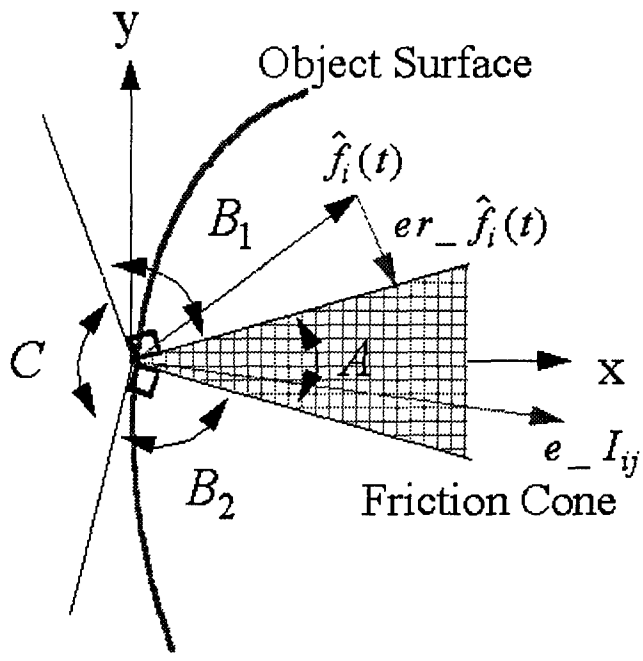

Figure 5.2 Definition of contact force zones

As before, $\mathbf{er\_\hat{f}}_i(t)$ is the contact force error associated with the current i^{th} contact force, $\hat{\mathbf{f}}_i(t)$. The contact force $\hat{\mathbf{f}}_i(t)$ is defined to lie in one of four zones, A, B_1, B_2, or C, as depicted in Figure 5.2. Each of these zones has a different definition of the contact force error, $\mathbf{er\_\hat{f}}_i(t)$. For the development which follows, assume $\hat{\mathbf{f}}_i(t) \in B_1$,

$$er\_\hat{f}_{ix}(t) = -\mu \frac{-\hat{f}_{iy}(t) + \mu \hat{f}_{ix}(t)}{1+\mu^2} \tag{5.11}$$

$$er\_\hat{f}_{iy}(t) = \frac{-\hat{f}_{iy}(t) + \mu \hat{f}_{ix}(t)}{1+\mu^2} \tag{5.12}$$

We may now expand $er\_dot_{ij}(t)$:

$$er\_dot_{ij}(t) = \left(e\_I_{ijx} \cdot er\_\hat{f}_{ix}(t) + e\_I_{ijy} \cdot er\_\hat{f}_{iy}(t)\right) / C_{dot} \tag{5.13}$$

$$er\_dot_{ij}(t) = \left(e\_I_{ijx} \cdot (-\mu \frac{-\hat{f}_{iy}(t) + \mu \hat{f}_{ix}(t)}{1+\mu^2}) + e\_I_{ijy} (\frac{-\hat{f}_{iy}(t) + \mu \hat{f}_{ix}(t)}{1+\mu^2})\right) / C_{dot} \tag{5.14}$$

$$er\_dot_{ij}(t) = \frac{1}{C_{dot}(1+\mu^2)}(\mu \widehat{f}_{ix}(t) - \widehat{f}_{iy}(t))(e\_I_{ijy} - \mu e\_I_{ijx}) \tag{5.15}$$

For this analysis, assume all contacts have the same value for μ, the static (Coulomb) coefficient of friction. Since the two components of the error vector are related as:

$$er\_\widehat{f}_{ix}(t) = -\mu \cdot er\_\widehat{f}_{iy}(t) \tag{5.16}$$

then we may simplify the square of the norm of the error,

$$er\_\widehat{f}_{ix}(t)^2 + er\_\widehat{f}_{iy}(t)^2 = (1+\mu^2) er\_\widehat{f}_{iy}(t)^2 \tag{5.17}$$

Substituting Equation 5.12 into Equation 5.17,

$$er\_\widehat{f}_{ix}(t)^2 + er\_\widehat{f}_{iy}(t)^2 = \frac{\left(-\widehat{f}_{iy}(t) + \mu \widehat{f}_{ix}(t)\right)^2}{1+\mu^2} \tag{5.18}$$

Similarly, both the local x and y components of the $(t+1)$ iteration of the i^{th} error in contact force are:

$$er\_\widehat{f}_{ix}(t+1) = dx_i(t+1) - \widehat{f}_{ix}(t+1) \tag{5.19}$$

$$er\_\widehat{f}_{ix}(t+1) = -\mu \frac{-\widehat{f}_{iy}(t+1) + \mu \widehat{f}_{ix}(t+1)}{1+\mu^2} \tag{5.20}$$

and

$$er\_\widehat{f}_{ix}(t+1) = \mu \frac{\widehat{f}_{iy}(t) + k(t) \cdot \sum_{j=1}^{m} w_{ij}(t) \cdot e\_I_{ijy} - \mu(\widehat{f}_{ix}(t) + k(t) \cdot \sum_{j=1}^{m} w_{ij}(t) \cdot e\_I_{ijx})}{1+\mu^2} \tag{5.21}$$

$$er\_\widehat{f}_{iy}(t+1) = \mu \cdot dx_i(t+1) - \widehat{f}_{iy}(t+1) \tag{5.22}$$

$$er\_\widehat{f}_{iy}(t+1) = \frac{-\widehat{f}_{iy}(t+1) + \mu \widehat{f}_{ix}(t+1)}{1+\mu^2} \tag{5.23}$$

and

$$er\_\widehat{f}_{iy}(t+1) = -\frac{\widehat{f}_{iy}(t) + k(t)\cdot\sum_{j=1}^{m} w_{ij}(t)\cdot e\_I_{ijy} - \mu(\widehat{f}_{ix}(t) + k(t)\cdot\sum_{j=1}^{m} w_{ij}(t)\cdot e\_I_{ijx})}{1+\mu^2} \tag{5.24}$$

Using Equation 5.17, the square of the norm of the $(t+1)^{th}$ iteration of the i^{th} contact force error is:

$$er\_\widehat{f}_{ix}(t+1)^2 + er\_\widehat{f}_{iy}(t+1)^2 = \frac{\left((\mu\widehat{f}_{ix}(t) - \widehat{f}_{iy}(t)) + k(t)\cdot\sum_{j=1}^{m} w_{ij}(t)(\mu e\_I_{ijx} - e\_I_{ijy})\right)^2}{1+\mu^2}. \tag{5.25}$$

Equations 5.18 and 5.25, which describe the contact force errors for the $(t+1)^{th}$ and t^{th} iteration, can be substituted into Equation 5.1

$$\sum_{i=1}^{m}(er\_\widehat{f}_{ix}(t+1)^2 + er\_\widehat{f}_{iy}(t+1)^2) - \sum_{i=1}^{m}(er\_\widehat{f}_{ix}(t)^2 + er\_\widehat{f}_{iy}(t)^2) < 0 \tag{5.26}$$

or equivalently,

$$\sum_{i=1}^{m}[(er\_\widehat{f}_{ix}(t+1)^2 + er\_\widehat{f}_{iy}(t+1)^2) - (er\_\widehat{f}_{ix}(t)^2 + er\_\widehat{f}_{iy}(t)^2)] = \sum_{i=1}^{m} dn_i(t) < 0 \tag{5.27}$$

where

$$dn_i(t) = \frac{2(\mu\widehat{f}_{ix}(t) - \widehat{f}_{iy}(t))k(t)\sum_{j=1}^{m} w_{ij}(t)(\mu e\_I_{ijx} - e\_I_{ijy})}{1+\mu^2} + \frac{k(t)^2(\sum_{j=1}^{m} w_{ij}(t)(\mu e\_I_{ijx} - e\_I_{ijy}))^2}{1+\mu^2} \tag{5.28}$$

One could enforce a more conservative convergence requirement than Equation 5.27 by requiring each $dn_i(t) < 0 \;\forall\; i$. This requirement means, for each contact, that the next iteration in contact force may not move away from the friction cone. In order to assure this, equations relating all contact errors must be formed and solved simultaneously. Since the FLRS does not consider all contacts simultaneously, convergence of each contact force at each step cannot be expected.

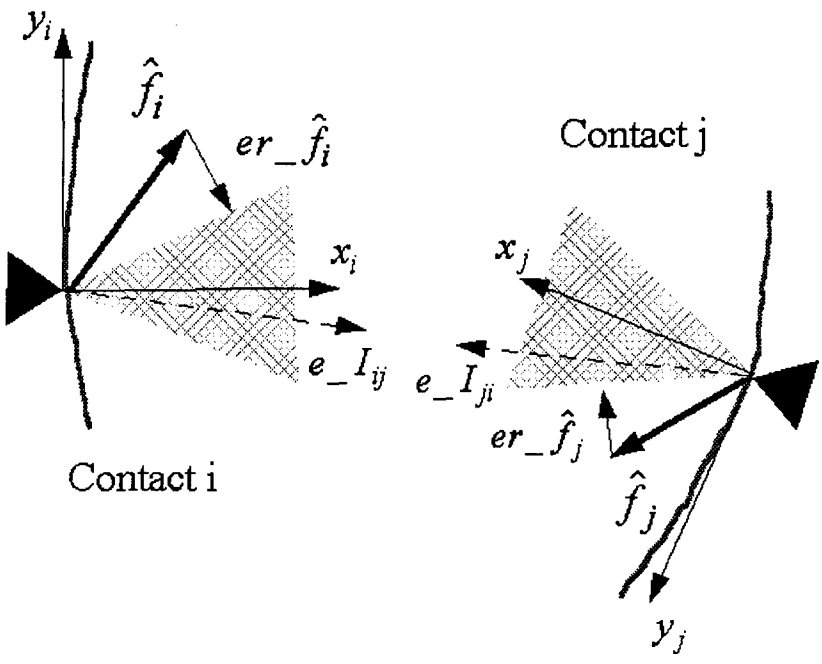

Figure 5.3 Two contact problem

5.1.1 Two Contact Problem. Let's now reduce the problem to $m = 2$, as illustrated by Figure 5.3. Assuming the less stringent convergence requirements of Equation 5.27, the sum of the difference of the contact force norms is:

$$\begin{aligned}\sum_{i=1}^{m} dn_i(t) = \frac{k(t)^2 \cdot w_{ij}(t)^2}{1+\mu^2}\left[(\mu e\_I_{ijx} - e\_I_{ijy})^2 + (\mu e\_I_{jix} - e\_I_{jiy})^2\right] + \\ \frac{2k(t) \cdot w_{ij}(t)}{1+\mu^2}\left[(\mu e\_I_{ijx} - e\_I_{ijy})\left(\mu \hat{f}_{ix}(t) - \hat{f}_{iy}(t)\right)\right] + \\ \frac{2k(t) \cdot w_{ij}(t)}{1+\mu^2}\left[(\mu e\_I_{jix} - e\_I_{jiy})\left(\mu \hat{f}_{jx}(t) - \hat{f}_{jy}(t)\right)\right]\end{aligned} \tag{5.29}$$

Forming the convergence requirement inequality and multiplying all terms by $1+\mu^2$ and dividing by $k(t)$,

$$k(t) \cdot w_{ij}(t)^2 \cdot a + 2w_{ij}(t) \cdot b < 0 \tag{5.30}$$

where

$$a = (\mu e\_I_{ijx} - e\_I_{ijy})^2 + (\mu e\_I_{jix} - e\_I_{jiy})^2 \tag{5.31}$$

and

$$b = (\mu e\_I_{ijx} - e\_I_{ijy})\left(\mu \hat{f}_{ix}(t) - \hat{f}_{iy}(t)\right) + (\mu e\_I_{jix} - e\_I_{jiy})\left(\mu \hat{f}_{jx}(t) - \hat{f}_{jy}(t)\right) \quad (5.32)$$

Substituting Equation 5.15 for the components in b,

$$b = -(1+\mu^2)C_{dot}(er\_dot_{ij}(t) + er\_dot_{ji}(t)) \quad (5.33)$$

Equation 5.31 is maximized if both $e\_I_{ijy} = -\mu e\_I_{ijx}$ and $e\_I_{jiy} = -\mu e\_I_{jix}$, resulting in:

$$a = (2\mu e\_I_{ijx})^2 + (2\mu e\_I_{jix})^2 \quad (5.34)$$

Also, this assumption requires $e\_I^2_{ijx} = \frac{1}{1+\mu^2}$ and $e\_I^2_{jix} = \frac{1}{1+\mu^2}$ which further simplifies Equation 5.34 to:

$$a = \frac{8\mu^2}{(1+\mu^2)^2} \quad (5.35)$$

Substituting these terms into Equation 5.30,

$$k(t) \cdot w_{ij}(t)^2 \cdot \frac{8\mu^2}{(1+\mu^2)^2} + 2w_{ij}(t)\left(-(1+\mu^2)C_{dot}\right)(er\_dot_{ij}(t) + er\_dot_{ji}(t)) < 0 \quad (5.36)$$

or

$$k(t) < \frac{(1+\mu^2)^3 C_{dot}}{4\mu^2 w_{ij}(t)^2} w_{ij}(t)\,(er\_dot_{ij}(t) + er\_dot_{ji}(t)) \quad (5.37)$$

The right hand side of Equation 5.37 is always positive since $w_{ij}(t)$ will always have the same sign as $(er\_dot_{ij}(t) + er\_dot_{ji}(t))$, from the definition of $w_{ij}(t)$.

5.1.2 General Multicontact Problem . Now, let's consider the general multicontact case where $m > 2$. Assuming the less stringent convergence requirements, of Equation 5.27,

$$0 > \frac{1}{1+\mu^2}\sum_{i=1}^{m} 2(\mu\hat{f}_{ix}(t) - \hat{f}_{iy}(t))k(t)\sum_{j=1}^{m} w_{ij}(t)(\mu e\_I_{ijx} - e\_I_{ijy}) + \frac{1}{1+\mu^2}\sum_{i=1}^{m} k(t)^2(\sum_{j=1}^{m} w_{ij}(t)(\mu e\_I_{ijx} - e\_I_{ijy}))^2 \quad (5.38)$$

We can form the equivalent to Equation 5.29 by multiplying all terms by $1+\mu^2$ and dividing by $k(t)$:

$$2\sum_{i=1}^{m}\sum_{j=1}^{m}(\mu\hat{f}_{ix}(t)-\hat{f}_{iy}(t))w_{ij}(t)(\mu e\_I_{ijx}-e\_I_{ijy})+k(t)\sum_{i=1}^{m}(\sum_{j=1}^{m} w_{ij}(t)(\mu e\_I_{ijx}-e\_I_{ijy}))^2 < 0 \quad (5.39)$$

which, after substituting Equation 5.15, becomes

$$-(1+\mu^2)C_{dot}2\sum_{i=1}^{m}\sum_{j=1}^{m} w_{ij}(t)\cdot er\_dot_{ij}(t) + k(t)\sum_{i=1}^{m}(\sum_{j=1}^{m} w_{ij}(t)(\mu e\_I_{ijx} - e\_I_{ijy}))^2 < 0 \quad (5.40)$$

Solving for the scale factor $k(t)$:

$$k(t) < \frac{(1+\mu^2)C_{dot}2\sum_{i=1}^{m}\sum_{j=1}^{m} w_{ij}(t)\cdot er\_dot_{ij}(t)}{\sum_{i=1}^{m}(\sum_{j=1}^{m} w_{ij}(t)(\mu e\_I_{ijx} - e\_I_{ijy}))^2} \quad (5.41)$$

So far we have only considered cases where $\hat{\mathbf{f}}_i(t)$ & $\hat{\mathbf{f}}_i(t+1)$ lie in region B_1 of Figure 5.2; for $\hat{\mathbf{f}}_i(t)$ & $\hat{\mathbf{f}}_i(t+1)$ lying in region B_2 the convergence criteria becomes:

$$2\sum_{i=1}^{m}\sum_{j=1}^{m}(\mu\hat{f}_{ix}(t)+\hat{f}_{iy}(t))w_{ij}(t)(\mu e\_I_{ijx}+e\_I_{ijy})+k(t)\sum_{i=1}^{m}(\sum_{j=1}^{m} w_{ij}(t)(\mu e\_I_{ijx}+e\_I_{ijy}))^2 < 0 \quad (5.42)$$

Solving for the scale factor $k(t)$:

$$k(t) < \frac{(1+\mu^2)C_{dot}2\sum_{i=1}^{m}\sum_{j=1}^{m} w_{ij}(t)\cdot er\_dot_{ij}(t)}{\sum_{i=1}^{m}(\sum_{j=1}^{m} w_{ij}(t)(\mu e\_I_{ijx} + e\_I_{ijy}))^2} \quad (5.43)$$

where

$$er\_dot_{ij}(t) = \frac{-1}{C_{dot}(1+\mu^2)}(\mu\hat{f}_{ix}(t) + \hat{f}_{iy}(t))(e\_I_{ijy} + \mu e\_I_{ijx}) \quad (5.44)$$

For the case of $\widehat{\mathbf{f}}_i(t)$ & $\widehat{\mathbf{f}}_i(t+1)$ lying in region C, the contact force error is:

$$\mathbf{er\_\widehat{f}}_i(t) = -\left\{ \begin{array}{c} \widehat{f}_{ix}(t) \\ \widehat{f}_{iy}(t) \end{array} \right\} \tag{5.45}$$

The square of the magnitude of the contact force errors for the t^{th} and $(t+1)^{th}$ iteration are:

$$er\_\widehat{f}_{ix}(t)^2 + er\_\widehat{f}_{iy}(t)^2 = \widehat{f}_{ix}(t)^2 + \widehat{f}_{iy}(t)^2 \tag{5.46}$$

and

$$\begin{aligned} er\_\widehat{f}_{ix}(t+1)^2 + er\_\widehat{f}_{iy}(t+1)^2 &= \widehat{f}_{ix}(t+1)^2 + \widehat{f}_{iy}(t+1)^2 \\ &= (\widehat{f}_{ix}(t) + k(t)\sum_{j=1}^{m} w_{ij}(t) e\_I_{ijx})^2 + \\ &\quad (\widehat{f}_{iy}(t) + k(t)\sum_{j=1}^{m} w_{ij}(t) e\_I_{ijy})^2 \end{aligned} \tag{5.47}$$

The convergence criteria equation, Equation 5.26, becomes:

$$\sum_{i=1}^{m}\left((\widehat{f}_{ix}(t) + k(t)\sum_{j=1}^{m} w_{ij}(t) e\_I_{ijx})^2 + (\widehat{f}_{iy}(t) + k(t)\sum_{j=1}^{m} w_{ij}(t) e\_I_{ijy})^2\right) - \sum_{i=1}^{m}\left(\widehat{f}_{ix}(t)^2 + \widehat{f}_{iy}(t)^2)\right) < 0 \tag{5.48}$$

which can be reformed into,

$$\begin{aligned} k(t)\sum_{i=1}^{m}\left((\sum_{j=1}^{m} w_{ij}(t) e\_I_{ijx})^2 + (\sum_{j=1}^{m} w_{ij}(t) e\_I_{ijy})^2\right) < \\ -2\sum_{i=1}^{m}\left(\sum_{j=1}^{m} w_{ij}(t) e\_I_{ijx} + \sum_{j=1}^{m} w_{ij}(t) e\_I_{ijy}\right) \end{aligned} \tag{5.49}$$

with which we may deduce

$$k(t) < \frac{-2\sum_{i=1}^{m}\sum_{j=1}^{m} w_{ij}(t)\cdot(\hat{\mathbf{f}}_i(t)\bullet \mathbf{e\_I}_{ij})}{\sum_{i=1}^{m}\left((\sum_{j=1}^{m} w_{ij}(t)e\_I_{ijx})^2+(\sum_{j=1}^{m} w_{ij}(t)e\_I_{ijy})^2\right)} \tag{5.50}$$

Of course, in general, any t^{th} iteration solution will have contact forces lying in all four regions. Assuming $\hat{\mathbf{f}}_i(t)$ & $\hat{\mathbf{f}}_i(t+1)$ lie in the same region, for any i, and the contacts lying in each region may be enumerated as:

$$i = \begin{cases} 1 \rightarrow m_A & \hat{\mathbf{f}}_i(t) \in A \\ m_A+1 \rightarrow m_{B1} & \hat{\mathbf{f}}_i(t) \in B_1 \\ m_{B1}+1 \rightarrow m_{B2} & \hat{\mathbf{f}}_i(t) \in B_2 \\ m_{B2}+1 \rightarrow m & \hat{\mathbf{f}}_i(t) \in C \end{cases} \tag{5.51}$$

then

$$\begin{aligned} k(t) < & \frac{(1+\mu^2)2C_{dot}\sum_{i=m_A+1}^{m_{B1}}\sum_{j=1}^{m} w_{ij}(t)\cdot er\_dot_{ij}(t)}{\sum_{i=m_A+1}^{m_{B1}}(\sum_{j=1}^{m} w_{ij}(t)(\mu e\_I_{ijx}-e\_I_{ijy}))^2} + \\ & \frac{(1+\mu^2)2C_{dot}\sum_{i=m_{B1}+1}^{m_{B2}}\sum_{j=1}^{m} w_{ij}(t)\cdot er\_dot_{ij}(t)}{\sum_{i=m_{B1}+1}^{m_{B2}}(\sum_{j=1}^{m} w_{ij}(t)(\mu e\_I_{ijx}+e\_I_{ijy}))^2} + \\ & \frac{-2\sum_{i=m_{B2}+1}^{m}\sum_{j=1}^{m} w_{ij}(t)\cdot(\hat{\mathbf{f}}_i(t)\bullet \mathbf{e\_I}_{ij})}{\sum_{i=m_{B2}+1}^{m}\left((\sum_{j=1}^{m} w_{ij}(t)e\_I_{ijx})^2+(\sum_{j=1}^{m} w_{ij}(t)e\_I_{ijy})^2\right)} \end{aligned} \tag{5.52}$$

Thus, we have established an upper limit on the overall scale factor $k(t)$. The method described in Chapter IV to calculate $k(t)$ must be checked to ensure it is less than $k(t)$ as calculated through Equation 5.52; alternatively, and more practically, one could monitor the $\sum_{i=1}^{m} dn_i(t)$ to ensure the sum is less than zero and only use Equation 5.52 to develop a suitable $k(t)$ when the convergence check fails. Next, we need to establish a lower bound on $k(t)$.

In Chapter IV, $k(t)$ was calculated as:

$$k(t) = \Delta\hat{\mathbf{f}}_L^{\#}(t)\mathbf{er\_f}_L(t), \tag{5.53}$$

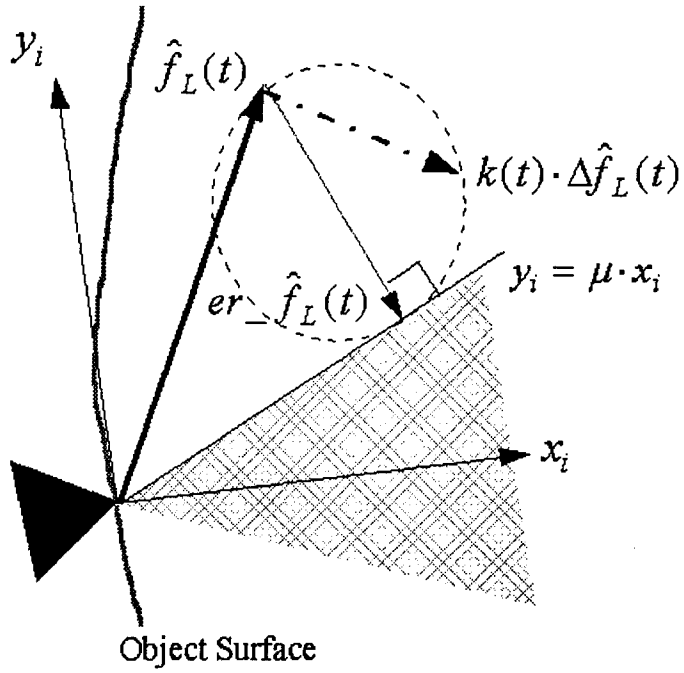

Figure 5.4 Relationship between $k(t)$, $\mathbf{er\_\hat{f}}_L(t)$, and $\Delta\hat{\mathbf{f}}_L(t)$

where L is the contact which has the greatest value of $\Delta\hat{\mathbf{f}}(t)\bullet\ \mathbf{er\_\hat{f}}(t)$. The superscript # indicates the pseudoinverse operator, $\mathbf{A}^{\#} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$. Since $\Delta\hat{\mathbf{f}}_L(t)$ is a vector, Equation 5.53 may be expanded to:

$$k(t) = (\Delta\hat{\mathbf{f}}_L^T(t)\mathbf{er\_\hat{f}}_L(t))/(\Delta\hat{\mathbf{f}}_L^T(t)\Delta\hat{\mathbf{f}}_L(t)). \tag{5.54}$$

For $\Delta\hat{\mathbf{f}}_L(t)\bullet\ \mathbf{er\_\hat{f}}_L(t) > 0 \Rightarrow k(t) > 0$ while for $\Delta\hat{\mathbf{f}}_L(t)\bullet\ \mathbf{er\_\hat{f}}_L(t) < 0 \Rightarrow k(t) < 0$. This overall scale factor provides a mechanism to ensure an appropriate magnitude of change for the contact forces. Assuming that the L^{th} contact is known, $k(t)$ scales the change in contact force for the L^{th} contact such that the norm of the difference between $k(t)\cdot\Delta\hat{\mathbf{f}}_L(t)$ and $\mathbf{er\_\hat{f}}_L(t)$ is minimized. A graphical representation of this solution is illustrated in Figure 5.4.

5.1.3 FLRS and the Optimal Solution. Both the FLRS algorithm and the optimal solution proposed by Nakamura begin with the external force solution, which

is itself the minimum norm of the contact force solution, without constraints. The FLRS seeks solutions whereby internal forces are added to any contact forces which violate the friction constraints. Ideally, for a given contact, FLRS adds internal forces which are coincident with a vector from the external force to the nearest point in force space which satisfies the friction constraints, i.e. perpendicular to the nearest point on the surface of the friction cone in Figure 5.4.

By defining $\mathbf{er\_\hat{f}}_i(t)$ as the shortest path to the constraint surface, and using the fuzzy control surface described in Chapter IV to calculate $w_{ij}(t)$, we are seeking a solution in which we attempt to use as little internal force as necessary to satisfy the frictional constraints. This is qualitatively the same goal the optimal solution seeks: the minimum internal force necessary to satisfy the frictional constraints. The ability of the FLRS to accomplish this goal is hampered by the fact that the solution is accomplished considering contacts pairwise while the optimal solution considers all contacts together. Thus, in general, one should expect the FLRS solution to be suboptimal with respect to the solution proposed by Nakamura.

We could change the character of the FLRS solution by altering the definition of $\mathbf{er\_\hat{f}}_i(t)$ and/or the function $w_{ij}(t)$. Appendix A documents how redefining $\mathbf{er\_\hat{f}}_i(t)$ and the FLRS exit criteria can enforce a given level of contact stability in the face of contact force disturbances. By changing the definition of $w_{ij}(t)$, we can further alter the character of the solution, this is discussed in Appendix A.

5.2 Summary

This chapter has developed limits on the FLRS parameters in order for the system to converge to a solution given a force closure grasp configuration and object wrench. The internal force weighting function, $w_{ij}(t)$ has been limited to one of an increasing function of er_dot_{ij} and er_dot_{ji}. The scale factor, $k(t)$, for the change in internal forces for the t^{th} iteration has an upper bound according to Equation 5.52 and a lower bound of zero. The nominal scale factor definition, from Chapter IV, has

a specific geometric relationship between $\mathbf{er\_}\hat{\mathbf{f}}_L(t)$ and $\Delta\hat{\mathbf{f}}_L(t)$. The FLRS solution is suboptimal, with respect to the solution proposed by Nakamura, and it may be altered by changing the rules/functions which govern its behavior.

VI. FLRS Implementation Refinements

The FLRS algorithm defined in Chapter IV and analyzed in Chapter V may be considered ideal. Since the FLRS solution may be characterized as a position control scheme, though the solution will converge, it may take an excessive number of iterations to do so. The initial implementation of the FLRS algorithm was accomplished using the MATLAB computational environment. During the implementation of the FLRS algorithm, two refinements were made related to solution acceleration. The first change concerned the redefinition of $\mathbf{er\_}\hat{\mathbf{f}}_i(t)$. The second refinement was the development of an alternative method, relative to the definition given in Chapter IV, to determine the internal force scale factor, $k(t)$.

6.1 Redefinition of Contact Force Errors

One absolute requirement of the FLRS algorithm is that no contact force may violate the friction constraints. Thus, we require a zero error solution using a method which may converge slowly near the constraint surface due to the position control characteristics of the solution. In order to speed the convergence of the FLRS solution, $\mathbf{er\_}\hat{\mathbf{f}}_i(t)$ is redefined to promote faster convergence. The new definition of $\mathbf{er\_}\hat{\mathbf{f}}_i(t)$ will be equivalent to shifting the friction cone of the i^{th} contact inward along the local x_i axis by a distance δx and will be called the convergence cone, as illustrated in Figure 6.1. The distance δx will be referred to as the convergence cone offset.

Referring to Figure 6.1, the new definition of the contact force error where $\hat{\mathbf{f}}_i(t) \in B_1$ will be:

$$\mathbf{er\_}\hat{\mathbf{f}}_i(t) = \left\{ \begin{array}{l} dx_i(t) \\ \mu \cdot (dx_i(t) - \delta x) \end{array} \right\} - \left\{ \begin{array}{l} \hat{f}_{ix}(t) \\ \hat{f}_{iy}(t) \end{array} \right\} \tag{6.1}$$

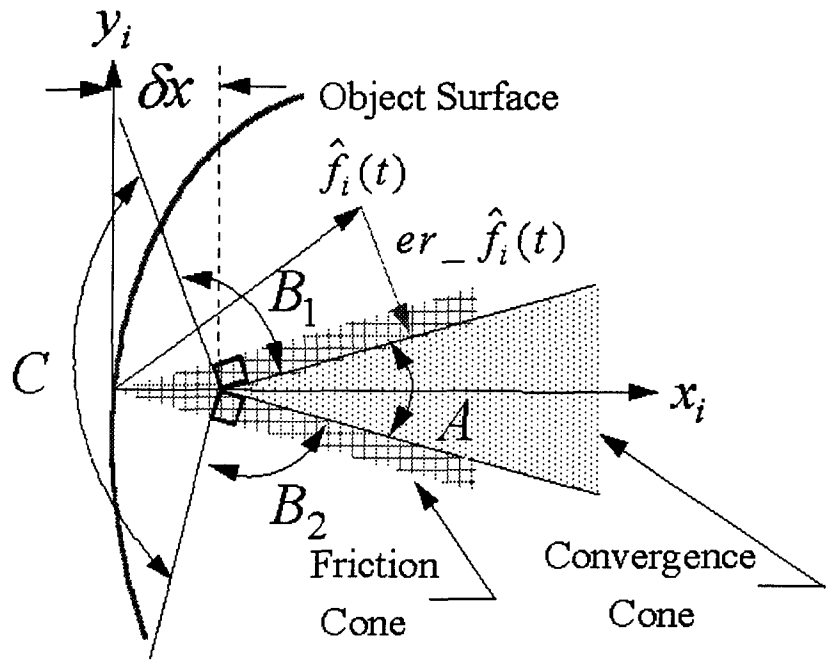

Figure 6.1 Definition of convergence cone

where

$$dx_i(t) = \frac{\hat{f}_{ix}(t) + \mu \cdot \hat{f}_{iy}(t) + \mu^2 \delta x}{1 + \mu^2} \tag{6.2}$$

For contact forces $\hat{\mathbf{f}}_i(t) \in B_2$,

$$\mathbf{er\_}\hat{\mathbf{f}}_i(t) = \begin{Bmatrix} dx_i(t) \\ -\mu \cdot (dx_i(t) - \delta x) \end{Bmatrix} - \begin{Bmatrix} \hat{f}_{ix}(t) \\ \hat{f}_{iy}(t) \end{Bmatrix} \tag{6.3}$$

where

$$dx_i(t) = \frac{\hat{f}_{ix}(t) - \mu \cdot \hat{f}_{iy}(t) + \mu^2 \delta x}{1 + \mu^2} \tag{6.4}$$

and for $\hat{\mathbf{f}}_i(t) \in C$,

$$\mathbf{er\_}\hat{\mathbf{f}}_i(t) = \begin{Bmatrix} \delta x \\ 0 \end{Bmatrix} - \begin{Bmatrix} \hat{f}_{ix}(t) \\ \hat{f}_{iy}(t) \end{Bmatrix} \tag{6.5}$$

Referring to Figure 4.1, the friction constraint satisfaction check made at the end of every iteration is still based on the friction cone, not the convergence cone. The

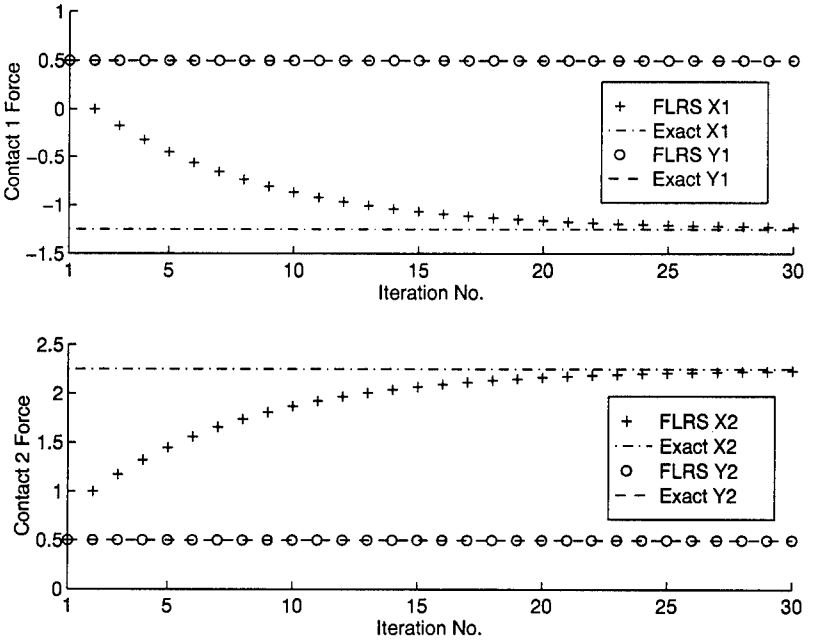

Figure 6.2 FLRS solution steps for two contact problem

effect of this change in the FLRS algorithm may be demonstrated using the two contact example of Chapter II.

Recall the problem and solution to the two contact example solved using the optimal method described by Nakamura:

$$W = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix}, \quad Q = \begin{Bmatrix} 1 \\ 1 \\ 0 \end{Bmatrix}, \; and \quad F = \begin{Bmatrix} -1.25 \\ 0.5 \\ 2.25 \\ 0.5 \end{Bmatrix} \tag{6.6}$$

Now examine the solution using the original FLRS algorithm without the new definition for $\mathbf{er\_}\widehat{\mathbf{f}}_i(t)$. Figure 6.2 illustrates the iterative development of the contact force components for both Contacts 1 and 2. Even after 30 iterations of the FLRS algorithm, the friction constraints have not been satisfied. It should be clear that the contact forces will not satisfy the constraints in a finite number of iterations.

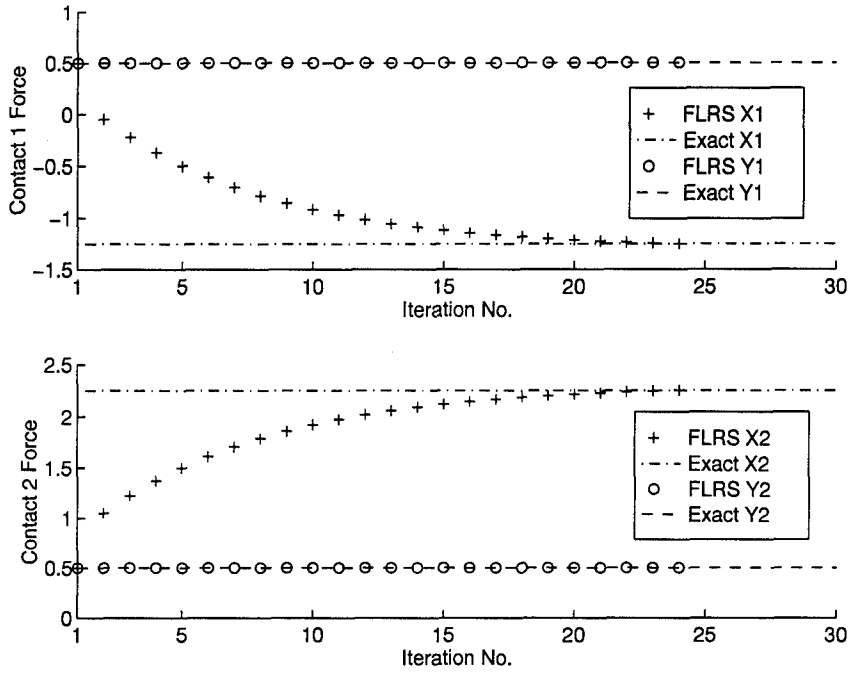

Figure 6.3 FLRS solution steps with convergence cone

Figure 6.3 illustrate the FLRS solution using the redefined $\mathbf{er\_\hat{f}}_i(t)$, where $\delta x = 0.05$. These solutions satisfy the frictional constraints after 24 of iterations of the FLRS algorithm. Solution sensitivity to the convergence cone offset will be addressed later.

6.2 Refined Scale Factor

Another situation leading to slowed solution convergence is the case where a single contact L, is repeatedly controlling solution convergence through the calculation of $k(t)$, as defined by Equation 5.54. This situation typically occurs when all but one or two contact forces satisfy the friction constraints, and $\Delta\mathbf{\hat{f}}_L(t) \bullet \mathbf{er\_\hat{f}}_L(t) \ll 1$. Referring to Figure 5.4, this situation arises when $\Delta\mathbf{\hat{f}}_L(t) \bullet \mathbf{er\_\hat{f}}_L(t) \ll 1$, or $\Delta\mathbf{\hat{f}}_L(t)$ is nearly perpendicular to $\mathbf{er\_\hat{f}}_L(t)$. Thus, $k(t)\Delta\mathbf{\hat{f}}_L(t)$ will be small and the solution will slowly converge. For these cases, a refined definition for $k(t)$ will allow a speedy

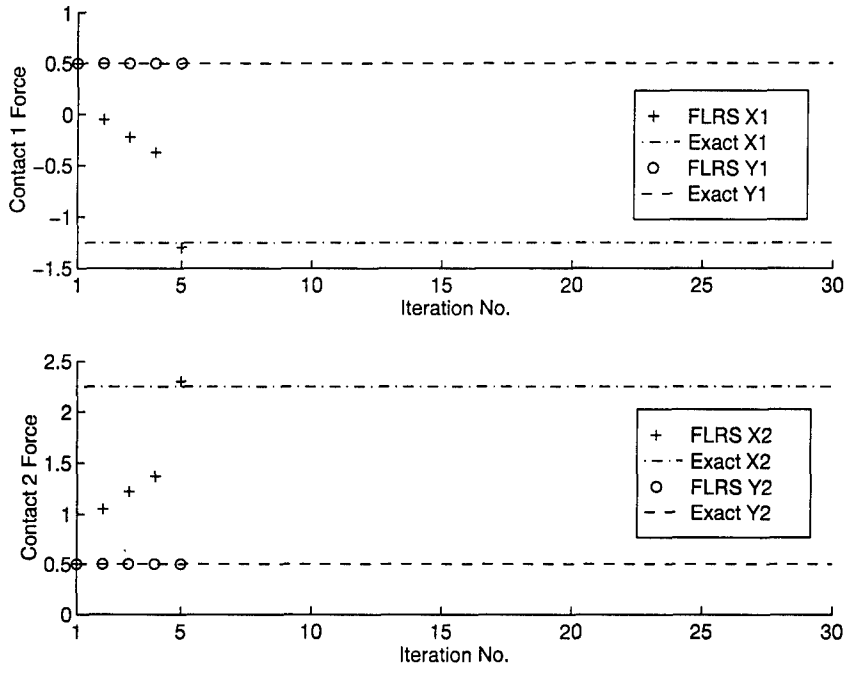

Figure 6.4 FLRS solution steps with convergence cone and $\bar{k}$

solution. This situation may be demonstrated using the two contact example of Chapter II.

Using the redefined $\mathbf{er\_\hat{f}}_i(t)$, where $\delta x = 0.05$, the FLRS algorithm exhibited the slow convergence behavior seen in Figure 6.3. Recognizing the repetitive nature of the solution, one could extrapolate a previous solution to determine a better value for the scale factor $k(t)$. Assume the FLRS algorithm has just calculated $k(t)$ using the same contact L for the past n iterations, where n is the iteration threshold above which FLRS will use an alternative method to calculate $k(t)$. Instead of using $k(t)$, we will extrapolate to determine a value $\bar{k}(t)$ which will cause $\left\|\mathbf{er\_\hat{f}}_L(t+1)\right\| = 0$. Let,

$$\bar{k}(t) = \frac{\left\|\mathbf{er\_\hat{f}}_L(t)\right\|^2}{\Delta\hat{\mathbf{f}}_L(t) \bullet \mathbf{er\_\hat{f}}_L(t)} \tag{6.7}$$

Figure 6.4 illustrates the change in FLRS solution behavior with the refined scale factor definition included. The solution converged in just five iterations.

These two refinements to the FLRS algorithm allow, what is essentially a proportion control law, to converge relatively quickly. Chapter VII explores numeric examples of FLRS solution sensitivity to changing δx and μ. Section A.4 discusses other means by which the FLRS convergence properties may be improved.

VII. FLRS Numeric Examples

Three grasp configurations were used to numerically validate the FLRS solution method. The FLRS algorithm is now assumed to include both the convergence cone and $\bar{k}$ amendments discussed in Chapter VI. The three configurations consist of three, four, and five contacts on a unit radius sphere. Each configuration was tested with 72 coplanar commanded object wrenches. Each solution was accomplished using the FLRS algorithm, the numerical equivalent to the Nakamura optimal solution, and a realistic approximation to the Nakamura optimal solution. The three contact case was also solved using the method proposed by Yoshikawa. Also, the five contact configuration was tested with 72 random spatial commanded object wrenches. The MATLAB code used to generate the FLRS solutions is included in Appendix B.

7.1 Grasp configurations

The five contacts used for the numeric examples are illustrated in Figure 7.1. The i^{th} column of the matrix **Pos** is the position vector of the i^{th} contact in the object frame.

$$\mathbf{Pos} = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0.707 & 0.707 \\ 0 & 0 & 0 & 0.707 & -0.707 \end{bmatrix} \tag{7.1}$$

The three, four, and five contact configurations applied contacts $\{1,2,3\}$, $\{1,2,3,4\}$, and $\{1,2,3,4,5\}$ respectively. The commanded object wrenches, for the first series of examples, coincides with the object (x,y) plane.

$$\mathbf{Q} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{7.2}$$

where $\theta = i * \pi/36$, and has units of degrees, and $i = 1:72$. Contacts $\{1,2,3\}$ lie in the plane of the commanded object wrenches.

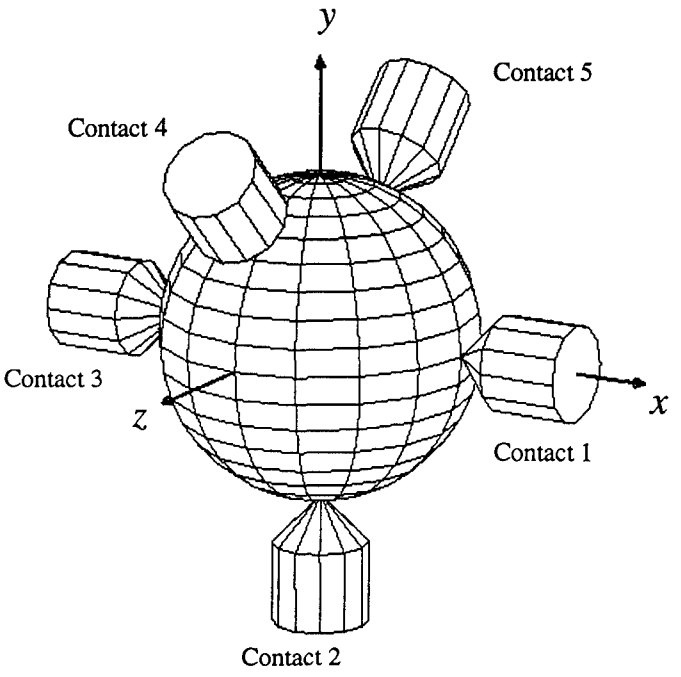

Figure 7.1 Example object and five contacts

7.2 Solution methods

All numeric examples were solved using three methods. The first method used the FLRS algorithm. The second consisted of a numeric approximation to the Nakamura optimal solution. The numeric approximation was accomplished using the MATLAB constrained optimization function *CONSTR* [1, 19]. This algorithm solves the Kuhn-Tucker equations, Equation 2.38 and 2.39, through the use of constrained quasi-Newton methods which guarantee superlinear convergence by accumulating second order information regarding the Kuhn-Tucker equations using a quasi-Newton updating procedure. These methods are commonly referred to as sequential quadratic programming (SQP) methods. A quadratic programming subproblem is solved at each major iteration which is then used to generate the search direction for a line search procedure. The CONSTR function may be given analytic gradients for the objective function and constraint functions, or the gradients may be numerically approximated.

The CONSTR function allows the user to set the acceptable error bounds on the independent variable, the constraint functions, and the objective function. For this case, the independent variable was the vector $\mathbf{y}$, the constraint functions, $\mathbf{g}(\mathbf{y})$, were those of Equation 2.34, and the objective function, $h(\mathbf{y})$, was $\mathbf{F}^T\mathbf{F}$. For the numeric approximation to the optimal solution, the error bounds for these elements were set very small, $\varepsilon_y = 0.001$, $\varepsilon_g = 0.00001$, $\varepsilon_h = 0.001$. However, even with the error bounds set very small, many of the solutions obtained through the *CONSTR* function actually violate the frictional constraints. For the purposes of these examples, we will ignore this fact. The figures below present this *CONSTR* solution as the theoretical Nakamura solution.

A practical analog to the above optimal solution was used to make relevant comparisons between the method proposed by Nakamura and the FLRS method. The realistic analog uses a displaced friction cone, similar to the one used for the FLRS convergence cone. Since errors exist in the numeric solution and the frictional constraints cannot be violated, the nominal frictional constraints used in the *CONSTR* function must be displaced inward from the actual constraints. This allows all solutions within the given error bounds to fall inside the actual constraint boundary. In an effort to achieve similar solutions between the realistic optimal method and the FLRS method, the nominal friction constraints were equivalent to the convergence cone of the FLRS solution. The friction constraints were shifted inward δx along the local x-axes.

With the nominal frictional constraints set, the error bounds on the independent variables and the objective function were relaxed to: $\varepsilon_y = 0.1$ and $\varepsilon_h = 0.1$. The error bound on the constraint functions, ε_g, was increased until solutions for $\mathbf{F}$, began to exceed the original friction constraint boundary. The same error bound was used for all examples, $\varepsilon_g = 0.027$. Thus, the practical Nakamura solution achieved admissible results, nominally the same results as the FLRS algorithm, while reducing

the number of iterations necessary to converge, by relaxing the error bounds. This solution will be referred to as the practical Nakamura solution.

The three contact problem is also solved using the method proposed by Yoshikawa. This method also uses the CONSTR optimization routine to solve for internal forces. As with the Nakamura solution, the Yoshikawa solution will have both a theoretical and practical solution. The optimization related error bounds on the Yoshikawa solutions will be the same as those for the Nakamura solutions for the sake of comparison.

As mentioned above, the CONSTR algorithm is capable of using analytic gradient input for both the objective function and constraint functions, with respect to the unknown variable $\mathbf{y}$. The CONSTR solution data included in this document were obtained using the default numeric approximation for the objective and constraint function gradients. Results obtained using analytic gradient information were not significantly different in terms of contact force solution or the number of FPOs required for the solution.

7.3 Planar Object Wrench Study

Figures 7.2, 7.4, and 7.6 illustrate how well the practical Nakamura and FLRS solutions followed the theoretical Nakamura solution. Both the practical Nakamura and the FLRS solutions appear to be close to the theoretical Nakamura solution such that one would conclude they approximate a least squares solution. Figure 7.2 also illustrates the Yoshikawa solution; for the most part, the theoretical solution lies on theoretical Nakamura solution. One area where there is some noticeable deviation is near $\theta = 90°$. This is a manifestation of the Yoshikawa definition of manipulating and grasping force directions which can generate asymmetric grasping and manipulative force directions for a symmetric problem.

The Euclidean norm of the contact forces is presented rather than all the components of all the contact forces, which would appear as almost identical plots

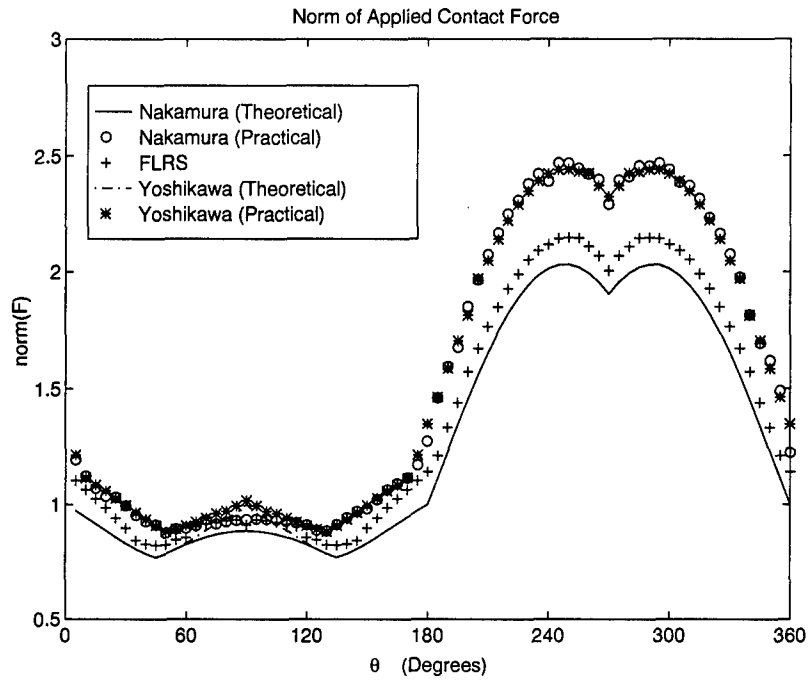

Figure 7.2 Norm($\mathbf{F}$) for three contact problem

between the three solution methods. The plot of $norm(\mathbf{F})$ actually exaggerates differences between the solutions. Figures 7.3, 7.5, and 7.7 illustrate the number of floating point operations (FPOs) required to achieve admissible solutions for both the practical Nakamura algorithm and the FLRS algorithm. The FPO count began just before external force calculation was performed until an admissible contact force solution was accomplished, for all methods. The pseudoinverse of the grasp matrix is required only once for a given grasp configuration for the methods illustrated, except the Yoshikawa solution which doesn't use one. The plots illustrate the dramatic decrease in required FPOs for the FLRS solution compared to the practical Nakamura solution.

Figure 7.8 illustrates the root mean square (RMS) of the FPOs for the 72 solutions for the three, four, and five contact problems. The results for the practical Nakamura case show the dramatic increase in required FPOs with an increase in the number of contacts. The FLRS case illustrates relative independence between the

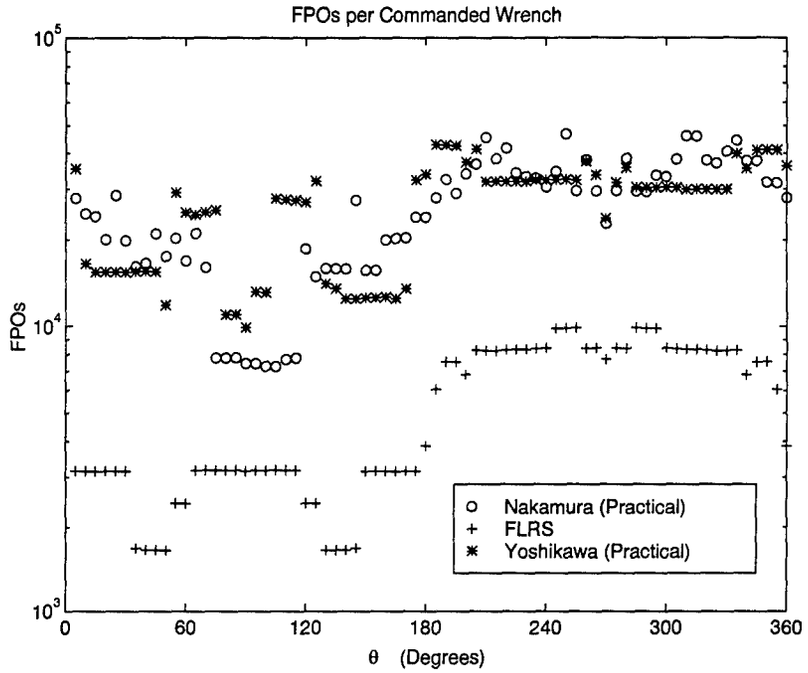

Figure 7.3 FPOs for three contact problem

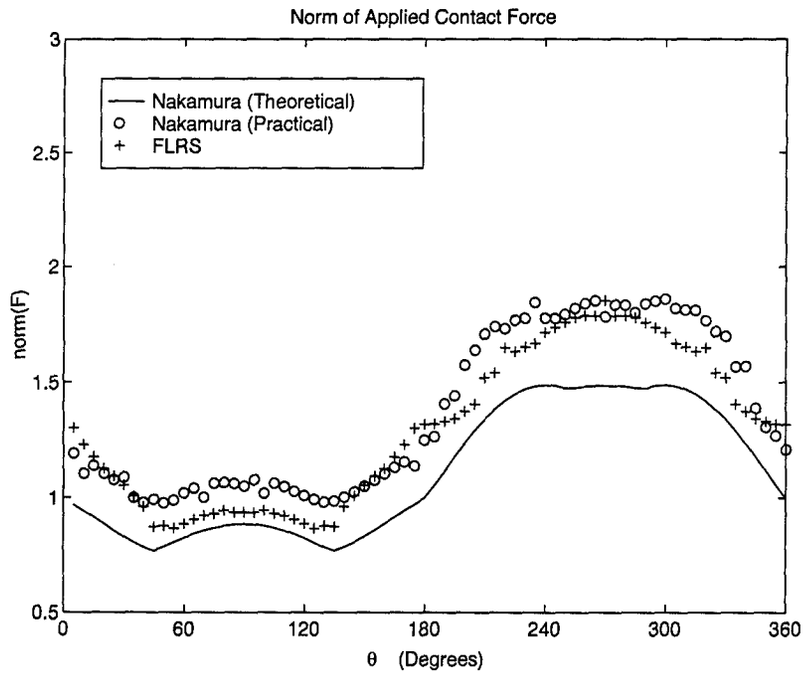

Figure 7.4 Norm($\mathbf{F}$) for four contact problem

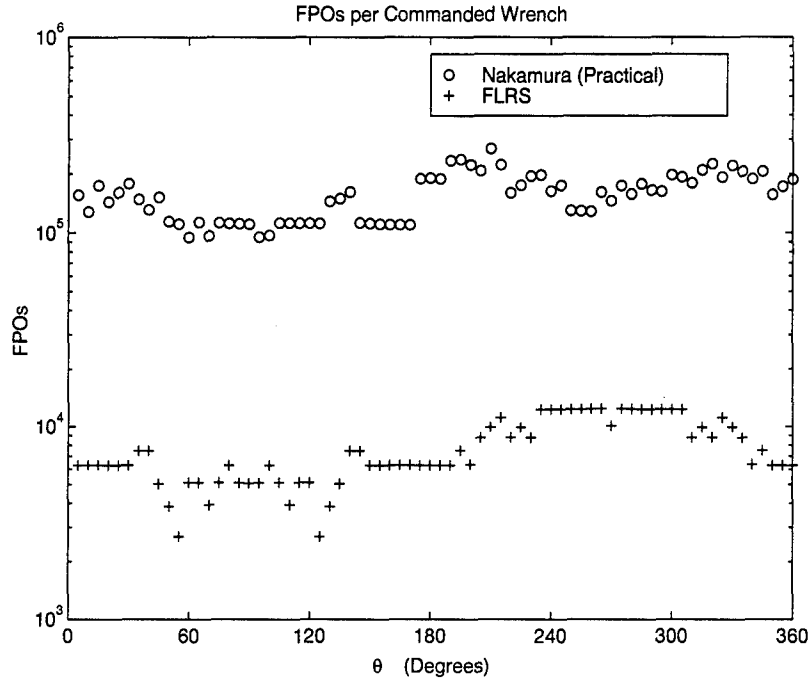

Figure 7.5 FPOs for four contact problem

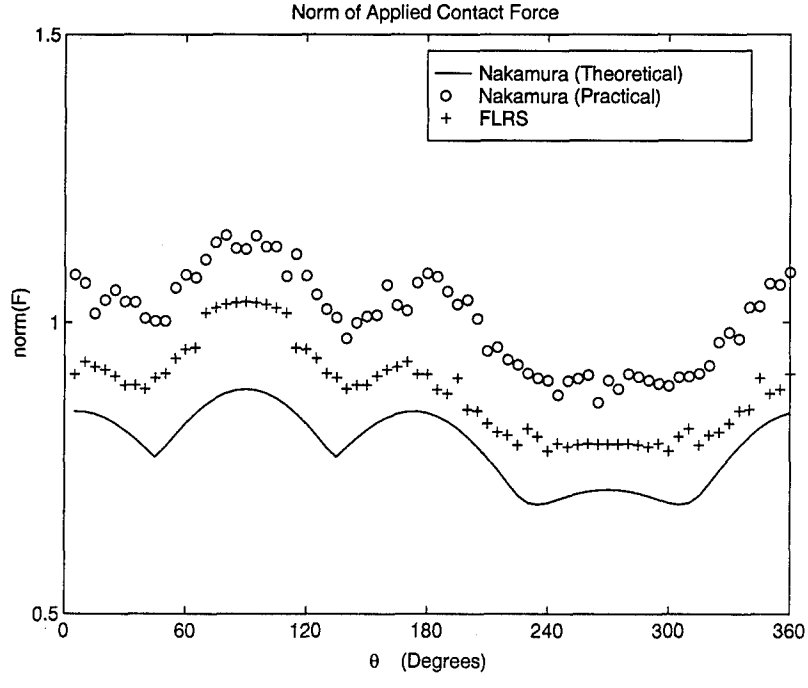

Figure 7.6 Norm($\mathbf{F}$) for five contact problem

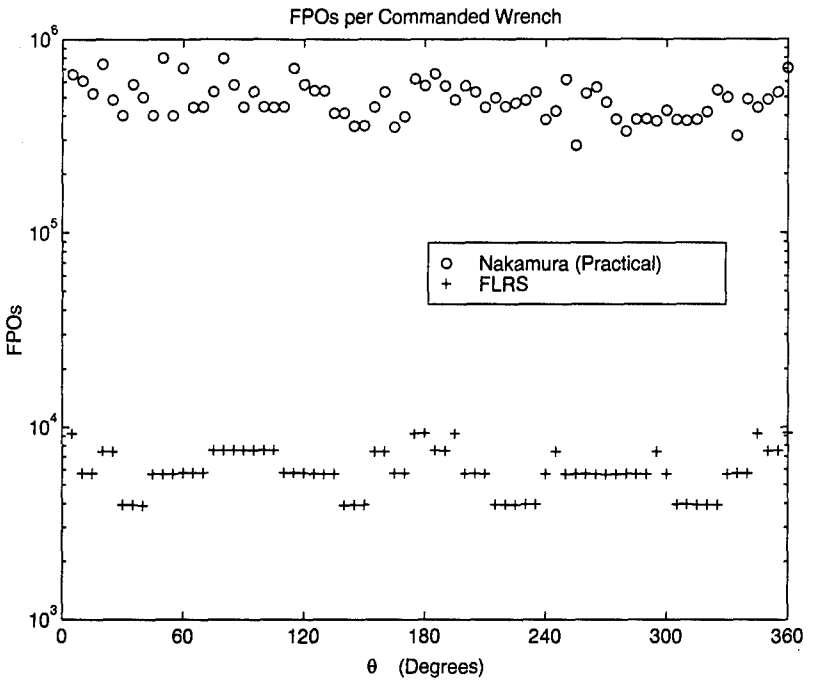

Figure 7.7 FPOs for five contact problem

required FPOs and the number of contacts. The Yoshikawa data point illustrates the degree to which his proposed solution accelerated the internal force solution process. The increase associated with the FLRS solution of the four contact problem is due, in part, to the asymmetry of the particular grasp configuration, relative to the commanded object wrenches. The ratio of RMS FPOs for the practical Nakamura solutions to the FLRS solutions are: 4.6, 19.7, and 81.3 for the three, four, and five contact problems respectively.

7.4 Random Object Wrench Study

The following numeric example used the five contact configuration illustrated in Figure 7.1. However, the commanded object wrenches consisted of 72 random spatial vectors where each element of the wrench was bounded, $Q_i \in [\ -1 \quad 1\ ]$. Figure 7.9 illustrates the norm($\mathbf{F}$) results for the commanded wrenches. The data was sorted such that the norm($\mathbf{F}$) for the theoretical Nakamura solution increases

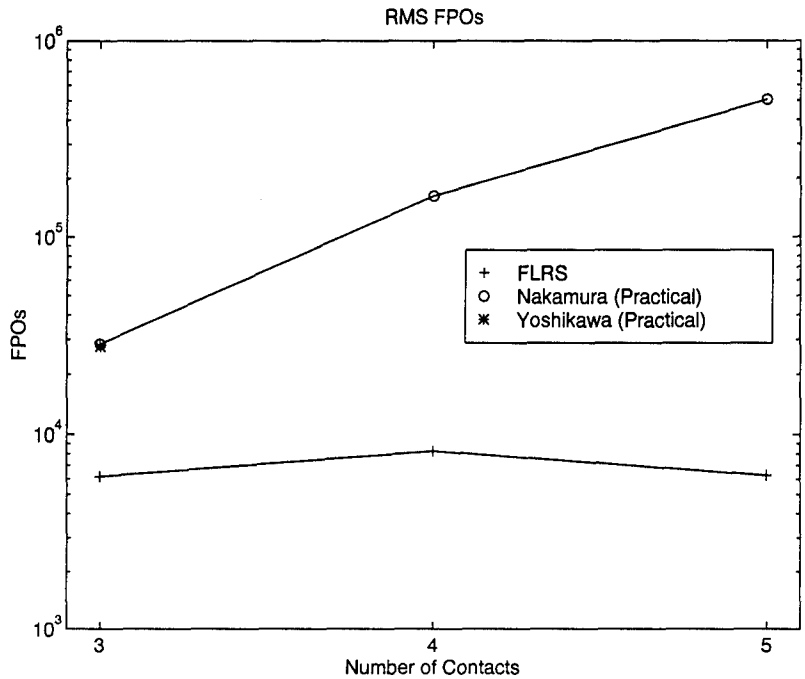

Figure 7.8 RMS FPOs for the three, four, and five contact problems

from the first commanded wrench to the last. This was accomplished to order to present coherent results rather than noise. Figure 7.10 presents the FPO data which again shows the superiority of the FLRS algorithm compared with the practical Nakamura solution, the RMS FPO ratio was 49.8. The FLRS algorithm did not fail to converge to any of the random object wrenches.

The illustrated results of the numeric examples clearly show the tremendous advantage in terms of FPOs of the FLRS algorithm over the practical Nakamura solution. The norm($\mathbf{F}$) plots show that both algorithms obtain contact force solutions close to the theoretical Nakamura solution. The advantage in FPOs translates into a significant computational advantage for real-time applications using the FLRS method as will be quantified in Section 7.7 which discusses a real-time implementation of the FLRS algorithm and the results.

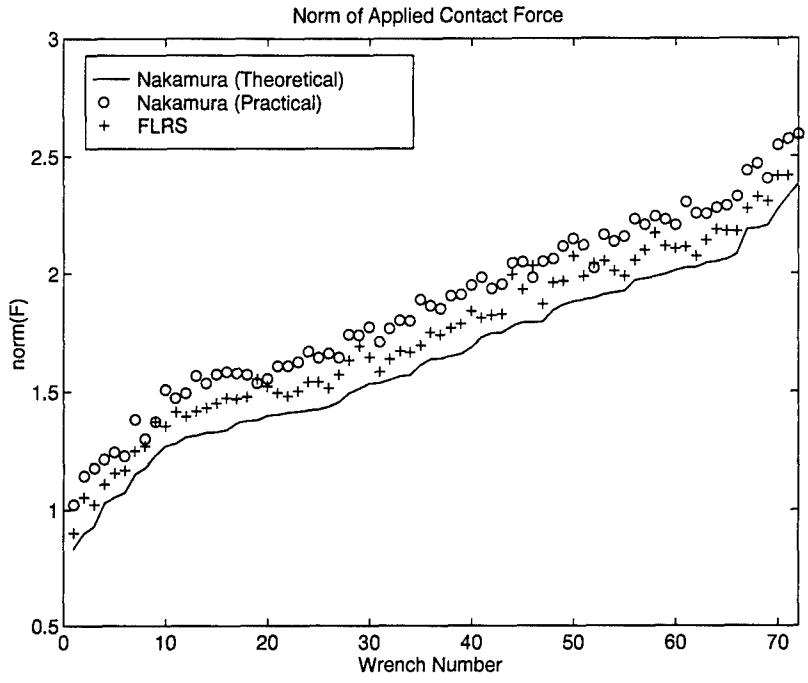

Figure 7.9 Norm($\mathbf{F}$) for the five contact problem with sorted random Q

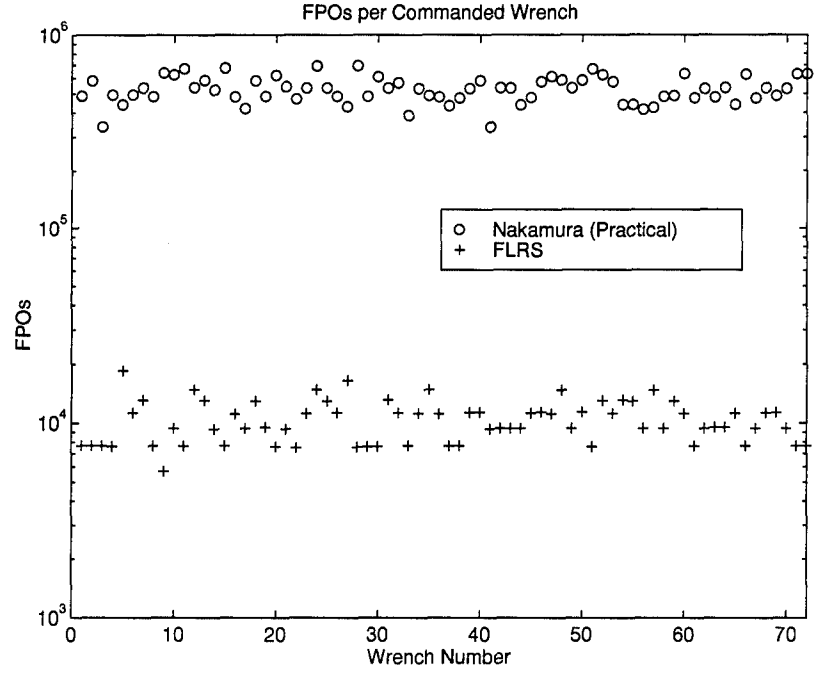

Figure 7.10 FPOs for five contact problem with sorted random Q

7.5 FLRS Convergence Cone offset Sensitivity

As mentioned earlier, the convergence cone offset, for the numeric examples illustrated, has been $\delta x = 0.05$. The fact that the convergence cone is used requires the contact force solutions, determined through the FLRS algorithm, to have greater levels of contact force relative to the theoretical minimum norm solution. This is apparent in all the previous figures. Clearly, larger convergence cone offsets will cause the FLRS solutions to have larger normed contact forces. Conversely, smaller convergence cone offsets will require a greater number of solution steps in order to converge to a solution. Clearly, a tradeoff exists between solution accuracy (based on the theoretical solution) and solution speed.

Figures 7.11 and 7.12 illustrate this tradeoff for the three contact example of Section 7.3. The ordinate of Figure 7.11 refers to the RMS of the difference between the norm of the contact forces calculated by the FLRS algorithm and the theoretical Nakamura method for 72 planar input wrenches. The input wrenches are the same as those used for the previous three contact example, Equation 7.2. Figure 7.12 illustrates the change in the RMS of the floating point operations required for each solution for progressively larger values of δx. The significant aspect of these figures is the knee of the curves, which is the basis for the decision to use $\delta x = 0.05$ as the nominal setting throughout this paper.

7.6 FLRS Coefficient of Friction Sensitivity

Up to this point, all examples have used a coefficient of friction, $\mu = 0.4$. Figure 7.13 illustrates the difference is contact force solutions for several values of μ. The three contact example of Section 7.3 is used for this illustration. Figure 7.14 illustrates a decrease in number of FPOs with an increase in μ. The RMS FPOs are 6160, 4127, and 3178 for $\mu = 0.4$, 0.5, and 0.6 respectively.

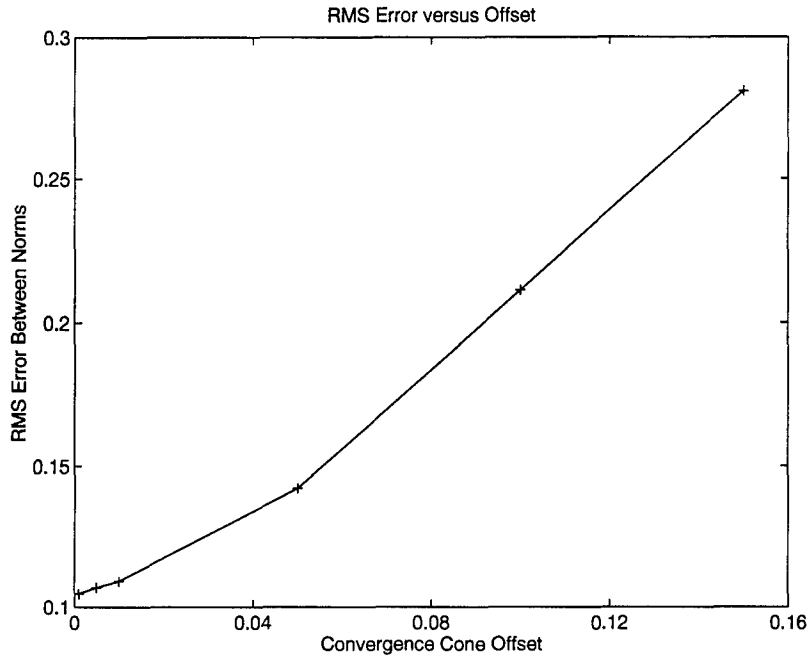

Figure 7.11 RMS Error for three contact problem

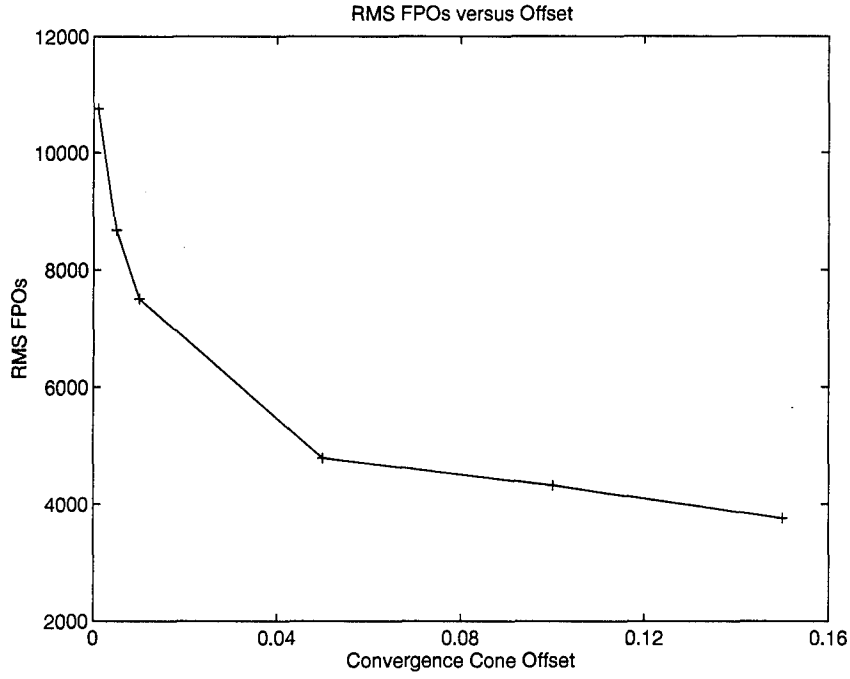

Figure 7.12 RMS FPOs for three contact problem

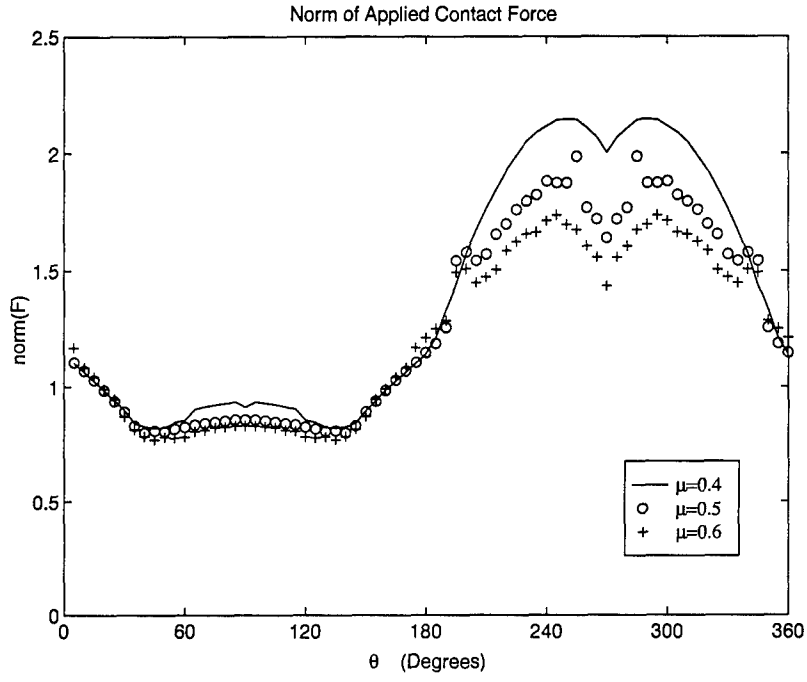

Figure 7.13 Norm($\mathbf{F}$) for three contact problem

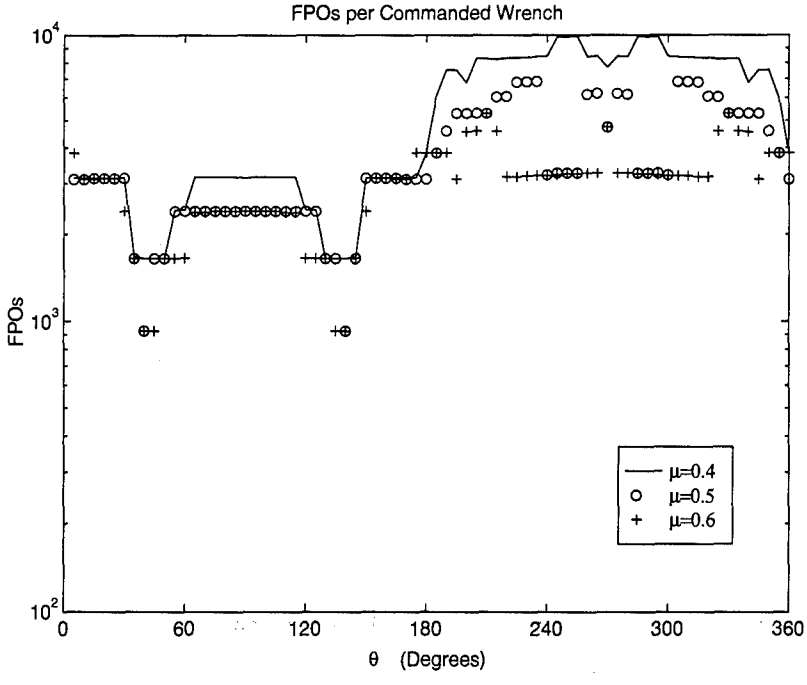

Figure 7.14 FPOs for three contact problem

7.7 Real-Time FLRS

The final step of the validation process involved a real-time evaluation of the FLRS algorithm. The FLRS algorithm was converted from the MATLAB interpretive environment to a common form of the C language. Gnu C is a common Unix based C compiler, and one which supports the Chimera real-time operating system. The Chimera 3.1 real-time operating system which was used for the validation was developed at Carnegie Mellon University (CMU).

Chimera resides on a host computer, which in this case is a Sun SPARCstation2, and provides control over at least one single board computer, or real-time processing unit (RTPU), installed in a VMEbus system. Chimera allows properly written and compiled modules, similar to conventional functions, to be downloaded from the host machine to the RTPU and executed. Chimera provides the necessary overhead to synchronize the execution of the modules and to exchange data between them through a common global state variable table. The RTPU used for the validation was a Motorola 68030 microprocessor.

The converted FLRS code was completed as a single Chimera module and is included in Appendix C. In order to test the module, an input/output module was written to provide the input data; grasp configuration, external force solution, and start time; and record the output data; contact forces and end time. The output contact forces were used to verify proper module execution while the difference in start and end times were used to characterize the real-time capability of the FLRS algorithm. The real-time test involved the three contact problem of Section 7.3. The 72 planar wrenches were consecutively input to the FLRS module and the results were recorded. Figure 7.15 illustrates the time required for the contact force solution for each of the input wrenches. The maximum time required for a solution was 58 milliseconds. This is fast, though improvements in speed may be accomplished by optimizing the FLRS code and taking advantage of the parallel aspects of the algorithm. Since the FLRS algorithm evaluates the internal force weight, $w_{ij}(t)$

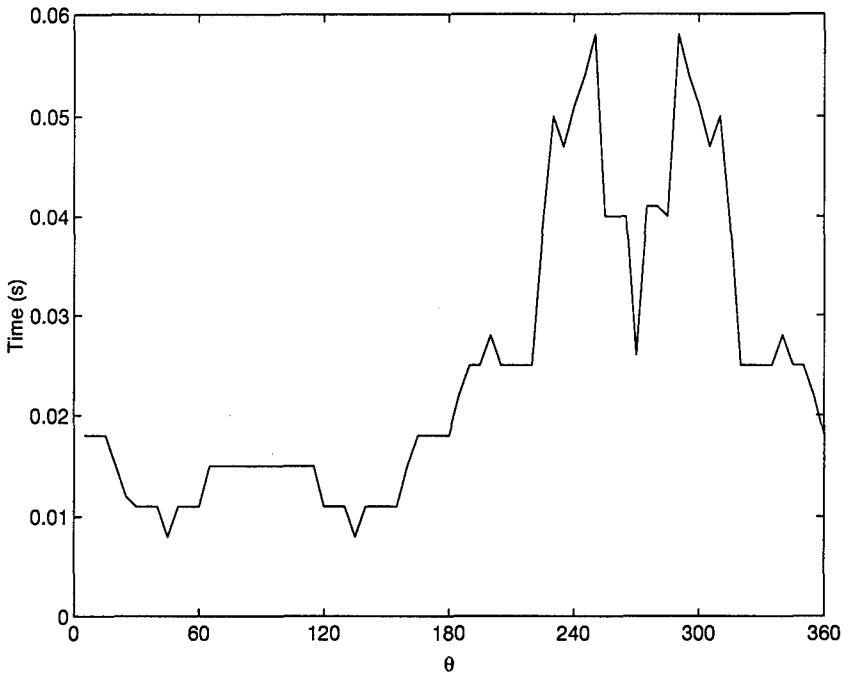

Figure 7.15 FLRS execution time for the three contact problem

independently for each internal force pair, a parallel implementation of the FLRS may reduce the execution time markedly.

7.8 Summary

The numeric examples illustrated the dramatic decrease in solution FPOs of the FLRS algorithm relative to the practical Nakamura method. The FLRS was also shown to achieve results very close to the optimal solution formulated by Nakamura. Finally, the FLRS algorithm was shown to be fast when implemented on real-time hardware.

VIII. Conclusion

This research explores a new method for calculating the contact forces required to stably manipulate a grasped object with frictional point contacts. Many authors in the field of robotic grasping have conducted extensive research in this area. One long standing problem in this area has been the determination of the static forces required to resist a given object wrench with a known contact configuration. A consistent theme throughout the published works is the need for a fast, reasonable, and flexible method to determine the internal forces for such a problem. The difficulty arises from the fact that, in general, an infinite number of valid solutions exist. Various definitions of optimal solutions have been proposed to limit the solution space. This research used the optimal definitions associated with Nakamura et al. His work has emphasized grasp force solutions which minimize the Euclidean norm of all the contact forces applied to the grasped object.

The goal of this research was to develop a fast, reasonable, and flexible method to determine the internal forces for a given grasp configuration and object wrench. Fast, means an algorithm capable of real time results required for successful robot grasping. Reasonable in this case refers to solutions qualitatively similar to the minimum Euclidean norm solutions proposed by Nakamura. Flexible is with regard to algorithm changes. One can easily change the character of the solution while still using the same fundamental algorithm. The algorithm which was developed, FLRS, accomplishes these tasks.

The FLRS algorithm is based on a tunable fuzzy logic inference method which can be altered to achieve changes in the character of the FLRS solutions. The rulebase used in this paper has emphasized solutions qualitatively similar to the minimum norm solutions of Nakamura. FLRS does not seek to simultaneously resolve appropriate internal force levels for a given contact force error state, as do published

optimization methods. Instead, FLRS weights changes in the internal forces on a pairwise basis. This results in a comparatively fast algorithm.

Based on a comparison with the Nakamura solution method, FLRS is significantly faster for a three contact problem and is dramatically faster for the four and five contact problems explored. The solutions are very close to the theoretical minimum norm solutions. In order to verify the real time capability, the three contact problem was accomplished using a compiled version of FLRS uploaded to a Motorola 68030 processor. The real time results were also good, with a mean of 24ms for each solution and a maximum of 58ms.

8.1 Contributions

This research has made a number of important contributions to the field:

- Established a floating point operation baseline of Nakamura's algorithm
- Differentiated between 'theoretical' and 'practical' problem formulations for optimal solutions
- The algorithm of Yoshikawa was shown to be suboptimal with respect to norm($\mathbf{F}$)
- The FLRS algorithm was established
  - Defined the contact force error
  - Evaluated the internal forces on a pairwise basis
  - Used a least squares internal force scale factor
  - Established an upper bounds on the scale factor to ensure solution convergence
  - Enhanced the convergence speed through the definition of a convergence cone and through scale factor extrapolation

- FLRS has dramatically decreased solution floating point operations relative to Nakamura's solution
- FLRS solution mimics the minimum norm solution proposed by Nakamura
- FLRS algorithm robustly solved problem of random spatial object wrenches

8.2 Recommendations

The FLRS algorithm is poised to enable strides in both simulation and hardware implementations of stable grasping algorithms. Currently, work is proceeding at AFIT which can use the FLRS algorithm as an integral part of the object resolved telerobotic (ORT) architecture for multifingered grasps. This work is partially an extension of the force control concepts of Nakamura. The ORT requires an algorithm which can calculate contact forces required to resist grasped object wrenches and provide contact stability. The FLRS algorithm, documented here, can fill this need.

One of the advantageous qualities of the FLRS algorithm is the ability to easily take advantage of parallel computer architecture. The FLRS algorithm operates on the contacts in an independent pairwise basis; this independence leads to a relatively easy implementation of simultaneous computation on parallel processors. The MATLAB and C code developed to date *has not* taken advantage of this possibility. Decreased execution time would be the incentive for such a change.

This research has not included significant exploration of the rulebase space and possible solution enhancements with changes in the rulebase. Typical methods for such exploration use automatic genetic algorithms seeking extremes of a given objective function. Also, the functional dependence of the convergence offset parameter with change in contact friction has not yet been fully characterized. These areas suggest opportunities for further research.

One final note concerning the physical implementation of a grasping robot. As mentioned in Chapter II, hardware requirements include high fidelity contact force sensors and force accuators. Without these elements, adequate control of the grasped object may not be possible. Limitations on contact force output from the grasping manipulators may be modeled in force space in addition to the friction constraint surfaces.

Appendix A. FLRS Extensions

This appendix documents some of the extensions to the FLRS algorithm. The first enhancement is the ability to smoothly engage and disengage contacts from a grasped object. This ability is crucial in order to accomplish grasp and regrasp of an object. The second extension is the adjustment of the contact friction cones in order to have some measure of robustness with regards to contact stability in the face of random or unexpected perturbations. Another extension involves mixed-mode contacts, where conventional point contacts with friction are used in conjunction with bi-directional contacts. Bi-directional contacts depend on structural load paths rather than friction constraint to maintain contact with the grasped object. The last extension discusses alternative definitions of the FLRS fuzzy rulebase used for weighting the internal forces.

A.1 Transitional Contacts

The FLRS algorithm may be modified to allow contact transitions that create force closure grasps with varying number of contacts. That is, contacts may be engaged and disengaged from the grasped object in a controlled manner. Recall,

$$\mathbf{Q} = \mathbf{WF} \tag{A.1}$$

where $\mathbf{F} = [\ \mathbf{f}_1 \quad \mathbf{f}_2 \quad \cdots \quad \mathbf{f}_m\ ]^T$ and $\mathbf{W}$ is the grasp matrix. For a given contact i, let $s\_i(t)$ denote the transition weight variable, $s\_i(t) \in [\ 0 \quad 1\ ]$. $s\_i(t)$ corresponds to the level of contact engagement. A contact that is fully engaged is defined to have $s\_i(t) = 1$, while a contact that is fully disengaged is defined to have $s\_i(t) = 0$. Let

$$\mathbf{S\_i}(t) = \begin{bmatrix} s\_i(t) & 0 & 0 \\ 0 & s\_i(t) & 0 \\ 0 & 0 & s\_i(t) \end{bmatrix} \tag{A.2}$$

We may now form the weighted grasp matrix as

$$\mathbf{Ws\_i}(t) = \begin{pmatrix} \mathbf{E}_3 & \cdots & \mathbf{S\_i}(t)\mathbf{E}_3 & \cdots & \mathbf{E}_3 \\ \mathbf{P}_1 & \cdots & \mathbf{S\_i}(t)\mathbf{P}_i & \cdots & \mathbf{P}_m \end{pmatrix} \tag{A.3}$$

Thus, for a given object wrench and i^{th} contact transition state, we may calculate the weighted external force vector as: $\mathbf{F}ext = \mathbf{Ws\_i}(t)^{\#}\mathbf{Q}$. The external forces so determined reflect a minimum norm solution of the weighted contact forces. This provides the basis for the FLRS algorithm to properly calculate the total transitional contact forces.

During the $\mathbf{er\_\hat{f}}_i$ calculation, the convergence cone offset, δx, for the i^{th} contact is redefined as: $\delta x s\_i(t) = s\_i(t) * \delta x$. Thus the convergence cone and $\mathbf{er\_\hat{f}}_i$ transition with the i^{th} contact. The last change to the FLRS algorithm is the redefinition of the change in internal force,

$$\mathbf{\Delta \hat{f}s_i}(t) = s\_i(t) * \sum_{j=1}^{m} w_{ij}(t) \cdot \mathbf{e\_I}_{ij} \tag{A.4}$$

this limits the use of internal forces associated with the i^{th} contact.

A contact transition is composed of n transition steps. Where n is the number of discrete solutions calculated during the transition. This value should depend on urgency of the transition, the robustness of the contact solutions (discussed later), and the force control capabilities of the manipulators in general. Each solution generated will be consistent with the overall FLRS algorithm, i.e. a minimum norm type solution for the rulebase described in this paper. During each step of the contact transition, the pseudoinverse of the weighted grasp matrix, $\mathbf{Ws\_i}(t)$, will be calculated. This additional calculation will thus slow the FLRS algorithm. However, if the number of transition steps is defined in advance, and the contact to be transitioned is known, all the pseudoinverse calculations may be accomplished in advance of the transition.

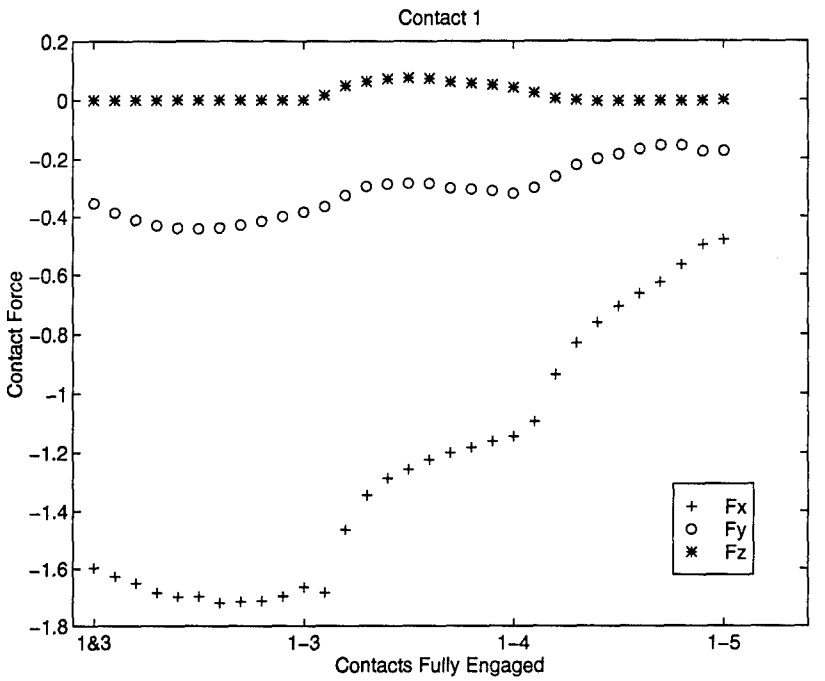

Figure A.1 Contact one, applied force during contact transition

A.1.1 Numeric Example. To demonstrate the transition capabilities of the FLRS, the five contact configuration of Figure 7.1 will be used. The example will begin with Contacts 1 and 3 providing the contact forces necessary to stably grasp the object and generate the object wrench, $\mathbf{Q} = [\ -.707 \quad -.707 \quad 0 \quad 0 \quad 0 \quad 0\ ]^T$. Contact 2 will be transitioned first to form a three contact grasp; likewise, Contacts 4 and 5 will be transitioned into contact with the object, in turn. At any given moment during the contact transitions, the object wrench remains the same. The solutions generated by the FLRS algorithm, during the transitions, will exhibit the standard minimum norm behavior as before. Ten steps will be used for each contact transition. Figures A.1-A.5 illustrate the change in contact force for all the contacts during the transitions.

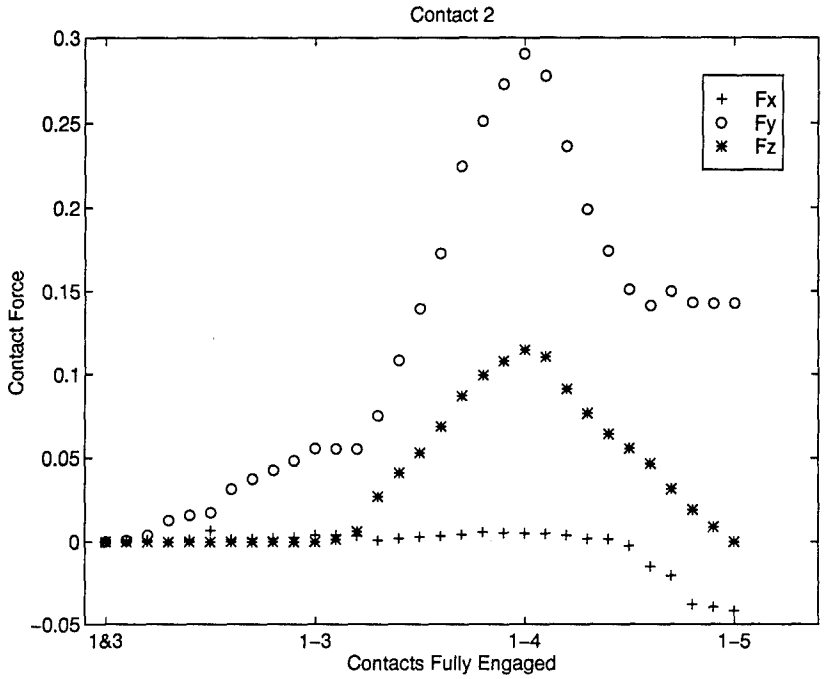

Figure A.2 Contact two, applied force during contact transition

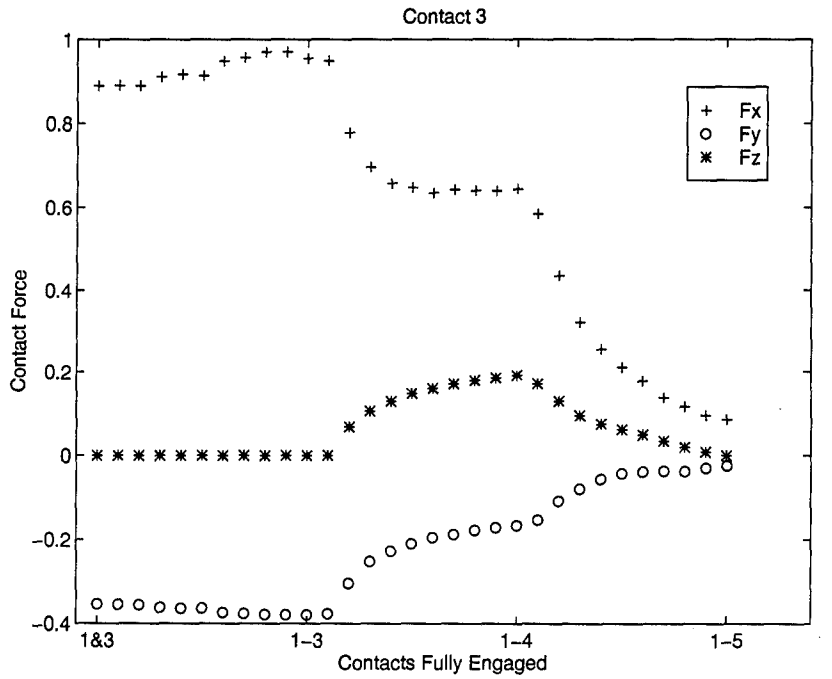

Figure A.3 Contact three, applied force during contact transition

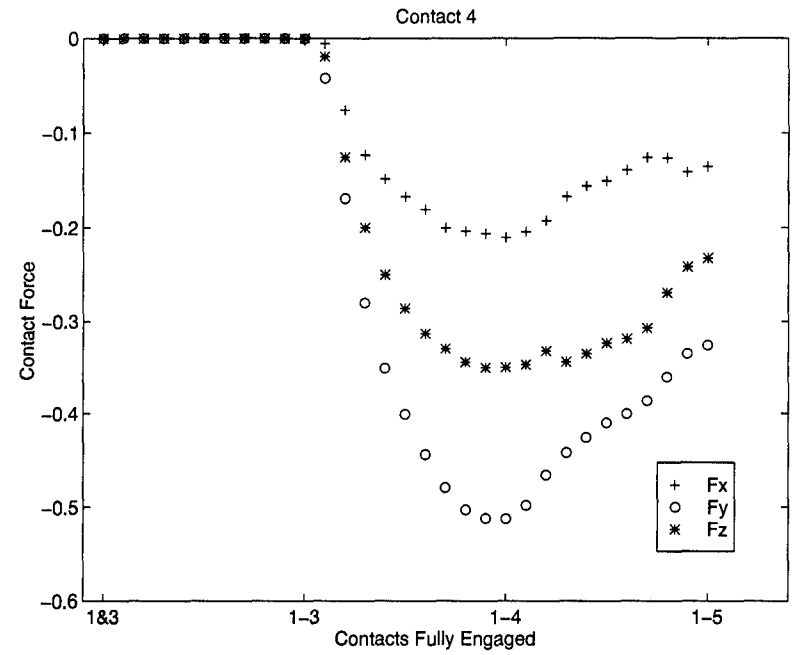

Figure A.4 Contact four, applied force during contact transition

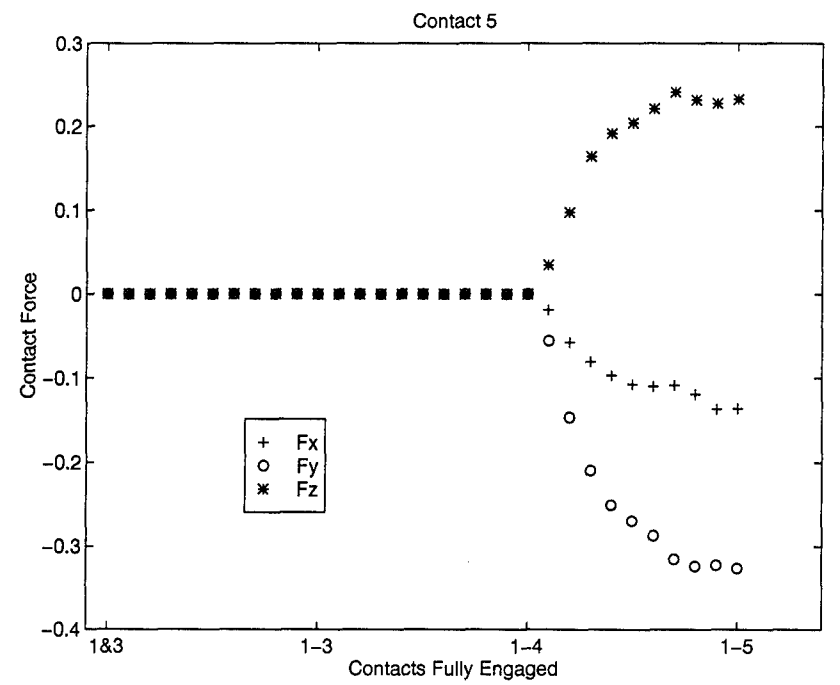

Figure A.5 Contact five, applied force during contact transition

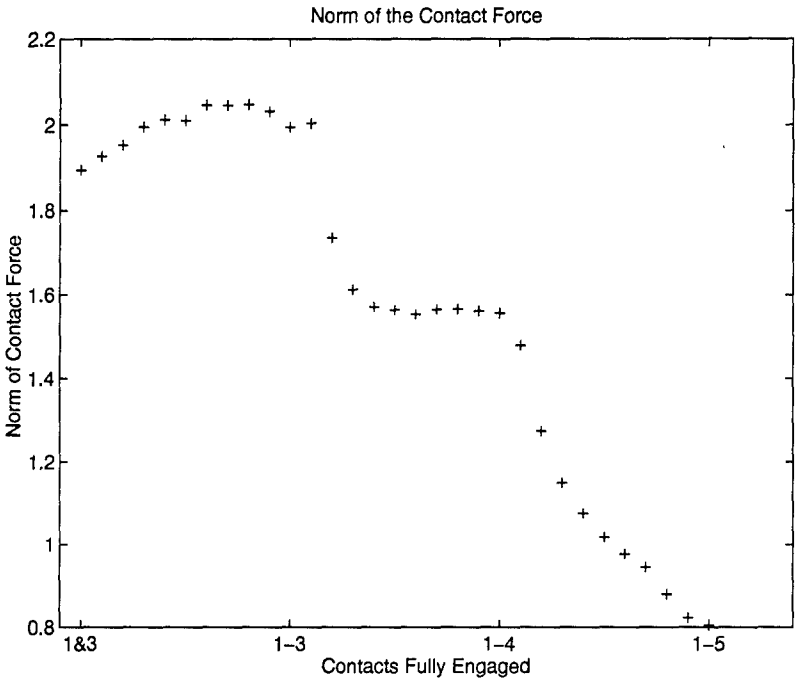

Figure A.6 Norm($\mathbf{F}$), during contact transition

A.2 Robust Contact Stability

During implementation of any control algorithm, system noise, from various sources, is present. Generally, designers must assume some characterization for the noise and design a control system which is stable in the face of such noise. The grasp force assignment algorithms will face similar challenges. Nakamura has developed equations which relate friction cone constraints with specified levels of contact stability [40]. A disturbance to the system is assumed to be an instantaneous acceleration of the grasped object. An equivalent contact disturbance, $\mathbf{f}_d$, may instead be assumed to exist and have a known maximum magnitude such that:

$$\|\mathbf{f}_d\| \leq a_c \tag{A.5}$$

Nakamura then forms a *contact-stability cone*, similar to the FLRS convergence cone. The analogous offset, the distance the contact-stability cone is moved inward along

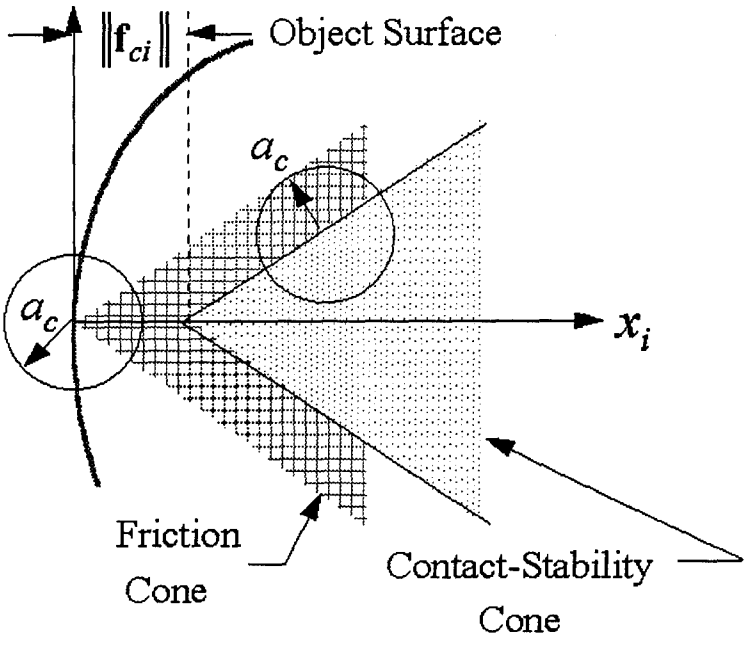

Figure A.7 Contact-stability cone

the local $\mathbf{x}_i$ axis, is

$$\|\mathbf{f}_{ci}\| = \frac{\sqrt{\mu^2+1}}{\mu} a_c \tag{A.6}$$

Figure A.7 illustrates the contact-stability cone. Thus disturbances, up to magnitude a_c, may be superposed on the contact force solution and the net contact force will still have contact stability. This development assumed a known constant maximum value of disturbance, independent of applied contact forces. If one wanted to also assume a linear relationship between actuator noise and actuation force, one could develop equations similar to Equations A.5 and A.6 but with an additional constraint on the cone angle of the contact-stability cone, as illustrated in Figure A.8. This enhanced contact-stability cone would then allow increasing levels of actuator noise with commanded actuator output, where $a_m = a_m(\mathbf{f}_i)$. The FLRS algorithm would substitute the contact-stability cone for the friction cone, in the previous development, and establish a convergence cone similar to the contact-stability cone but displaced a further δx along the local $\mathbf{x}_i$ axis.

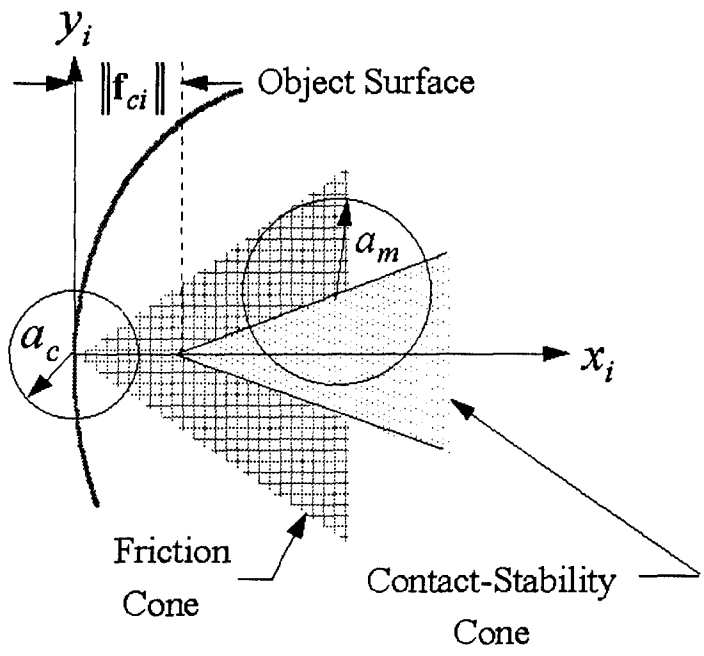

Figure A.8 Contact-stability cone with actuator disturbance

A.3 Mixed-mode Contact Grasps

Mixed-mode contact grasps are comprised of both conventional point contacts with friction constraints, and bi-directional contacts. Bi-directional contacts depend on structural load paths rather than friction constraint to maintain contact with the grasped object. Thus they are contacts which do not need any internal forces, and in fact should not have any for a minimum norm solution. The grasp force assignment solution proceeds in the same way the standard solution of Chapter IV, except that for any bi-directional contact, i, $\mathbf{er\_\hat{f}}_i(t) = 0$ and $er\_dot_{ij} = 0 \; \forall \, j$. Thus, any non-zero internal force pair associated with a bi-directional contact is the result of internal force requirements of a conventional point contact associated with the internal force pair.

A.4 Alternate FLRS Rulebase

The FLRS algorithm uses a fuzzy inference method for the weighting function, w_{ij}. Chapter V showed that convergence of the FLRS algorithm is dependent on the use of an appropriate weighting function; one could use an analytic function of the type

$$w_{ij}(t) = \frac{(er\_dot_{ij} + er\_dot_{ji})^n}{2^n} \tag{A.7}$$

where n is a positive odd integer. However, current research has proven that through the use of a fuzzy proportional controller, proportional-plus-derivative response, for a second order system can be obtained [45]. Thus, one can seek higher performance solutions with proportional data than a conventional proportional control law would allow. The fuzzy logic rulebase, and associated antecedent and consequent sets, allow for many degrees of freedom. Thus, one may seek varying solution behaviors by changing the many parameters which govern the solution. Typically, one would specify a desired behavior in terms of an objective function and use a genetic algorithm to determine a suitable fuzzy controller.

Appendix B. MATLAB Script

This appendix documents the MATLAB script that was used to generate the data presented in this document. This includes the code necessary to generate object wrenches and obtain solutions through the FLRS algorithm. The code is rough. The main code is: *main_fnl.* This code calls other script files and MATLAB functions, some were developed for this research while others are standard MATLAB functions. Any standard MATLAB functions will not be included in this appendix. Inside *main_fnl* are a number of flags which control solution and display modes. Hopefully, anyone with the fortitude of an ox will be able to determine what the options are and how to access them.

B.1 main_fnl

```
%this is the example contact force assignment method
%This code allows individual weighting of contacts,
%which allows contact transitions, and can be used to
%determine minimum friction cone offsets for a given set
%of numerical optimization termination criteria.  This
%code will also allow mixed mode contacts (uni-directional and
%bi-directional).  The capability to capture the d_F data as produced
%will allow analysis and graphical presentation of the solution
%process. Also capable of calling the Nakamura solution

% Mark Hunter, 25 Feb 96, Air Force Institute of Technology, WPAFB, OH.

clear

global T_i n in mu  mu2 eta del_X fc_x dX e_I e_Ic num_I RB_ef c_dot
cfd_wt afd_wt e_n_oi o_flag CT BC F_IT E_IT numit Ap_flag Fp_flag r Q
```

```
% if Ap_flag is on (1), plot graphical contact force convergence process
Ap_flag=0;
% if Fp_flag is on (1), plot F(i) vs i contact force convergence process
Fp_flag=0 ;

% coeffecient of friction
mu=.4;
mu2=mu^2;
mu3=(1+mu2)^3/(2*mu2);
eta=1/sqrt(1+mu^2);

% call contact initialization routine
cf_ini; %contact information

del_X=ones(n,1)*.05;   %friction cone displaced in local x-dir for
%convergence reasons
fc_x=ones(n,1)*0; %additional friction cone displacement for
%robust stability

% maximum and minimum normal force, not used yet
%Fn_max=4;
%Fn_min=.1;

% call grasp initialization routine
gf_ini

% initialize variables

in=1:n;
```

```
num_I=(n*n-n)/2; %number of internal forces

% define the rule-bases with which the fuzzified contact information
% will be compared with

RB_ef= [ 1 1 1 2 2 3 4; %1-Neg Lrg
1 2 2 3 3 4 5; %2-Neg Med
1 2 4 4 4 5 6; %3-Neg Sml
2 3 4 4 4 5 6; %4-Zero
2 3 4 4 4 6 7; %5-Pos Sml
3 4 5 5 6 6 7; %6-Pos Med
4 5 6 6 7 7 7]; %7-Pos Lrg

I6=eye(6);

A=null(W); %MATLAB 'null' command, calculates orthonormal basis
%for null of W, used in CONSTR solver
b=(3*n)-rank(W);
x_old=ones(b,1)/b;  %used in CONSTR solver as initial guess

%define bi-directional contacts, BC(i)=(1 or 0)
%thus the contact can support forces in all directions but no moments

BC=ones(n,1); %default setting is to have all contacts uni-directional
nct=10; %number of steps to transition a contact
% begin input data cycle

%for it=1:1 %number of solutions sought for this run
for it=1:72
```

```
it
%pause
%for ait=1:6 %use with force closure test
%for sit=1:2 %use with force closure test
%it=(ait-1)*2+sit; %use with force closure test

%define input object wrench
   th=(it)*pi/36;
   f_net=[cos(th) sin(th) 0];
% f_net=[1 1 0];
   m_net=[0 0 0];
   Q=[f_net m_net]';
%random wrench test
% Q=(rand(6,1)-.5)*2;



%calculate the 12 manipulation unit vectors, force closure test
%if sit==1
% Q=I6(:,ait);
%else
% Q=-I6(:,ait);
%end

%define contact transition vector, n elements  0<=CT(i)<=1
CT=ones(n,1); %default setting is to have all contacts fully engaged
% CT(5)=(it-1)/nct; %transition contact two
% CT(4)=0;
% CT(5)=0;

Wt_W=W; %copy W into Wt_W to allow for weighted grasp matrix
```

```
dX=del_X+fc_x; %del_X is the convergence buffer, fc_x is the
%stability offset

if min(CT)<1 %then at least one contact is transitional
for i=1:n
  if (CT(i)<1)
Wt_W(:,(i-1)*3+1:i*3)=Wt_W(:,(i-1)*3+1:i*3)*CT(i);
dX(i)=dX(i)*CT(i);
end %if
end %for
end %if

%pi_flops=flops; %zero counter

Ff=pinv(Wt_W)*Q; %get psuedoinverse solution (external forces)
Fo=Ff;
%pause
%pi_flops=flops-pi_flops %report number of flops for psuedoinverse

flops(0); %zero flops counter

flrs_flg=1;

if flrs_flg==1 %calculate FLRS solution

% Fh=f_sol(Ff,Cf,wt_it); %call fuzzy solver as script

f_sol; %call fuzzy solver as function
Fh=F;
```

```
%if algorithm plot switch, Ap_flag, is on,
%plot the contact force convergence;
if Ap_flag==1
a2_plot(F_IT,numit,E_IT);
end %if

%if  contact force history plot switch is on, do it
if Fp_flag==1
fp_plot(F_IT,numit);
[mi,ni]=size(F_IT);
nn=1:ni;
fs=[nn' F_IT'];
% save dx05q_2c.dat fs -ascii -tabs
end

% Fh is a 3 x n matrix of contact forces
h_fpos=flops; %record FPOs

end %flrs_flg

%save data to files for analysis
if flrs_flg==1
hm_fpos(it)=h_fpos/1.e6;
for i=1:n
F_h(i*3-2:i*3,1)=Fh(:,i);
end
% Qh_app(:,it)=W*F_h;
   Ch_save(:,it)=F_h;
```

```
end
if l_flag==1
nlm_fpos(it)=nl_fpos/1.e6;
Cnl_save(:,it)=Fn_l;
end

end %end input loop

if flrs_flg==1
save diss_h5 Ch_save hm_fpos %Q_com
% save dis_vmu4 Ch_save hm_fpos
end
```

B.1.1 cf_ini.m.

```
% this is the contact initialization routine for the main_fl.m script

% degrees to radians
deg2rad=pi/180;

% establish object base coordinates
x=[1 0 0]';
y=[0 1 0]';
z=[0 0 1]';

% enter number of contacts
%n=2;
n=3;
%n=4;
%n=5;
```

```
%assume object is unit sphere
rho=1;

% looking a the grasped object  model,
%the x-z plane forms the zero meridian
% azimuth(theta) (j) measured from z-axis
% the z-x plane forms the equator with
%positive elevation up towards y-axis,
% or North pole.
% units in degrees

%azm=[90 -90];
azm=[ 90 0 -90]; %r&a96 3 contacts
%azm=[ 90 0 -90 0]; %r&a96 4 contacts
%azm=[ 90 0 -90 0 180]; %r&a96 5 contacts

%ele=[0 0];
ele=[0 -90 0]; %r&a96 3 contacts
%ele=[0 -90 0 45]; %r&a96 4 contacts
%ele=[0 -90 0 45 45]; %r&a96 5 contacts

%azm=[ 90 -90 0 0];
%ele=[ 0  0 90 -90];
% convert units to radians

azm=azm*deg2rad;
ele=ele*deg2rad;

% calculate contact positions in x-y-z coordinates of r
```

```
for i=1:n
r(1,i)=rho*sin(azm(i))*cos(ele(i));
r(2,i)=rho*sin(ele(i));
r(3,i)=rho*cos(azm(i))*cos(ele(i));

% contact normal(contact x-axis) in
%object frame, for a sphere is easy
% (negative unit vector of position vector, r)
e_n_oi(:,i)=-r(:,i)/norm(r(:,i));

% determine appropriate tangent vectors to contact surface

e_t(:,1)=cross(e_n_oi(:,i),x);
e_t(:,2)=cross(e_n_oi(:,i),y);
e_t(:,3)=cross(e_n_oi(:,i),z);

for j=1:3
norm_e_t(j)=norm(e_t(:,j));
end

i_et=max(find(norm_e_t==max(norm_e_t)));
e_t_1(:,i)=e_t(:,i_et)/norm(e_t(:,i_et));

e_t_2(:,i)=cross(e_t_1(:,i),e_n_oi(:,i));
e_t_2(:,i)=e_t_2(:,i)/norm(e_t_2(:,i));


T_i(3*i-2:3*i,1:3)=...
[e_n_oi(:,i) e_t_1(:,i) e_t_2(:,i)];
```

```
end
```

B.1.2 gf_ini.m.

```
% Mark Hunter, 25 Feb 96, Air Force Institute of Technology, WPAFB, OH.
% this code defines the fuzzy sets and domains.
% also defines internal force unit vectors and grasp matrix
clear W
%vectors of center of triangular functions for input domain
c_dot=[-1 -.6 -.2 0 .2 .6 1]';
%c_dot[Ability of Internal force to affect error_force]
%plot_mem(c_dot, 'Internal.Error');

%vector of fuzzy singletons for Sugeno type output domain
cfd_wt=[-1 -.75 -.35 0 .35 .75 1]';
%cfd_wt=[-Lrg_Wt -Med_Wt -Sml_Wt No-Wt Sml_Wt Med_Wt Lrg_Wt]

% hardwire weights(areas) of cfd_wt
afd_wt=[1 1 1 1 1 1 1]';

d2r=pi/180;   %degree to radian conversion

e_y=[0 1 0]'; %establish global basis vectors
e_x=[1 0 0]';
e_z=[0 0 1]';

% establish internal force unit vectors
%and their weights associated with
% the fuzzy membership functions

for i=1:n
```

```
for j=1:n
   jj=(i-1)*n+j;
   if i==j
e_I(:,jj)=[0 0 0]';
else
      r_ij=r(:,i)-r(:,j);
      e_I(:,jj)=-r_ij/norm(r_ij); %internal force unit vectors
  end
e_Ic(:,jj)=T_i(3*i-2:3*i,1:3)'*e_I(:,jj);
%e_I in appropriate contact frame
end
end

% begin initialization for pseudo-inverse solution/external force solution
I3=eye(3,3);

for i=1:n
eval(['P' int2str(i) '=[0 -r(3,' int2str(i) ') r(2,' int2str(i) ');
r(3,' int2str(i) ') 0 -r(1,' int2str(i) ');
-r(2,' int2str(i) ') r(1,' int2str(i) ') 0 ];'])
end

for i=1:n
eval([ 'Wi=[I3;P',int2str(i),'];' ]);
W=[W Wi];
end

Wp=pinv(W);

%end initialization routine
```

B.1.3 f_sol.m.

```
%This is the heart of the FLRS algorithm
%function F=f_sol(Fo,Cf,wt_it)
%global T_i n in mu mu2 mu3 eta del_X fc_x dX
%e_I e_Ic num_I RB_ef c_dot cfd_wt afd_wt e_n_oi
%o_flag CT BC F_IT E_IT numit Ap_flag Fp_flag
iq=zeros(3,1);
n_max=30; %maximum number of iterations allowed
%n_max=25;  %for display purposes

numit=0; %initialize number of iterations until solution
if Ap_flag==1 | Fp_flag==1
%record Fc so that the change in contact
% force may be graphically displayed
F_IT=zeros(3*n,n_max);
E_IT=zeros(3*n,n_max);
end

C=1; %initialize scale factor C
C_old=1;

   for i=1:n
      F_e(:,i)=Fo((i-1)*3+1:(i*3));
   end

% determine the error between current commanded contact force
% and actual possible contact force.  Use shortest path to the friction
% cone as the basis for the error.

F_i_old=zeros(3,n);
```

```
% calculate current contact forces
F(:,in)=F_e(:,in)+F_i_old(:,in);

for i=1:n
Fc(:,i)=T_i(3*i-2:3*i,1:3)'*F(:,i); %convert to local
  %frame
theta(i)=atan2(Fc(3,i),Fc(2,i));  %angle closest to
  %original F_e
     end
%Fc
numit=numit+1;
if Ap_flag==1  | Fp_flag==1
%record Fc so that the change in contact force may be graphically
  %displayed
for ic=1:n
F_IT(ic*3-2:ic*3,numit)=F(:,ic);
end
end
% is any current contact force outside of friction cone
for i=1:n
fc_check(i)=-e_n_oi(:,i)'*F(:,i)+eta*norm(F(:,i));
end

% begin iterative loop to find solution
while (max(fc_check)>0 & (numit-1)<n_max)%contact
  %stability is violated

% determine what the goal vector should be based on the direction
% of F_e
```

```
for i=1:n
% calculate error based on current F and goal
if fc_check(i)<=0
ER_fc(:,i)=zeros(3,1);
else

% calculate closest point on friction cone from present contact force.
dx_p=(Fc(1,i)+mu*( Fc(2,i)*cos(theta(i)) +...
 Fc(3,i)*sin(theta(i))...
)+mu2*dX(i) )/(1+mu2);

% make any dx_*<dX equal to dX
dx=dx_p*(dx_p>=dX(i))+dX(i)*(dx_p<dX(i));

ER_fc(:,i)=[dx
(dx-dX(i))*mu*cos(theta(i))
(dx-dX(i))*mu*sin(theta(i))]-Fc(:,i);

end
%convert to global frame
ER_f(:,i)=T_i(3*i-2:3*i,1:3)*ER_fc(:,i);
nER_f(i)=norm(ER_f(:,i));
    end

if Ap_flag==1  | Fp_flag==1
%record ER_f so that the change in contact force may be
  %graphically displayed
for ic=1:n
E_IT(ic*3-2:ic*3,numit)=ER_f(:,ic);
end
```

```
end

% loop through all the internal forces
%
wt_tot=zeros(1,n*n);
ER_dot=zeros(n*n,1);

for i=1:n
for j=1:n
jj=(i-1)*n+j;
% calculate the dot product between the contact error and the
% internal force for each contact
% ER_dot(jj)=dot(e_Ic(:,jj),ER_fc(:,i));
%do this in local frame
ER_dot(jj)=dot(e_I(:,jj),ER_f(:,i));
%try this in global frame, shouldn't make any difference

% if the finger being faded in or out is i, then weight the
% associated ER_dot's
ER_dot(jj)=ER_dot(jj)*CT(j)*BC(j);
end
end

%determine max value of ER_dot

ER_dot_m=max(max(abs(ER_dot)));
ER_dot_m=ER_dot_m*(ER_dot_m>0)+(ER_dot_m==0);

%normalize ER_dot
ER_dot=ER_dot/ER_dot_m;
```

```
tst_flag=0; %test the use of analytic function rather than fuzzy

if tst_flag==0
% calculate memberships of the ER_dot scalars
[mf_ERd,mu_ERd]=mem_edci(ER_dot,c_dot);

% now apply the rule base to the input data to weight the
% individual internal forces.
wt_ef=zeros(7,num_I);
% wt=zeros(7,num_I);
I_count=0;
for i=1:n-1
for j=1:n
if i~=j & i<j
jj=(i-1)*n+j;
ii=(j-1)*n+i;
I_count=I_count+1;

for l=1:2
for k=1:2
i_wt_mf=RB_ef(mf_ERd(jj,l),mf_ERd(ii,k));
wt_ef(i_wt_mf,I_count)=wt_ef(i_wt_mf,I_count)+...
min(mu_ERd(jj,l),mu_ERd(ii,k));

end
end

end
end
end
```

```
%wt_ef
wt_tot=(((afd_wt.*cfd_wt)'*wt_ef)./(afd_wt'*wt_ef));
%wt_tot
%define symetric internal force weighting matrix
I_count=0;
for i=1:n-1
for j=1:n
if i~=j & i<j
I_count=I_count+1;
w(i,j)=wt_tot(I_count);
w(j,i)=wt_tot(I_count);
end
end
end
%w

else %tst_flag~=0
%this is a trial method to determine what interaction the internal
%forces should have without using fuzzy logic
kk=0;
I_count=0;
for i=1:n-1
for j=1:n
if i~=j & i<j
I_count=I_count+1;
jj=(i-1)*n+j;
ii=(j-1)*n+i;

if(nER_f(i)+nER_f(j))==0
wt_tot(I_count)=0;
```

```
else
d_er=ER_dot(jj)+ER_dot(ii);
wt_tot(I_count)=d_er/(2+1/abs(d_er)^kk)*...
(2+.5^kk)/2;
end
end
end
end
%[ER_dot wt_tot']

end  %end tst_flag

d_F=zeros(3,n);
I_count=0;
for i=1:n-1
for j=1:n
if i~=j & i<j
I_count=I_count+1;
jj=(i-1)*n+j;
ii=(j-1)*n+i;
d_F(:,i)=d_F(:,i)+wt_tot(I_count)*e_I(:,jj);
d_F(:,j)=d_F(:,j)+wt_tot(I_count)*e_I(:,ii);
end
end

end

% keep track of last three contacts which have
% had greatest dot(d_F,ER_f)
iq(3)=iq(2);
```

```
iq(2)=iq(1); %update last iq
%find contact corresponding to the largest dot(d_F,ER_f)
chk=0;
for i=1:n
dF_ER(i)=dot(d_F(:,i),ER_f(:,i));
if dF_ER(i)>chk
chk=dF_ER(i);
iq(1)=i;
end
end

%keep track of past iq,ER_f(:,iq), and d_F(:,iq)
%if iq doesn't change,

if numit>3 & iq==[iq(1) iq(1) iq(1)]'
c_flag=6;
% c_flag=2;
else
c_flag=2;
end
% determine C, in order to acheive least squares solution (min norm)
if c_flag==1
C=pinv(d_F(:,iq(1)))*ER_f(:,iq(1));

% since inputs to pseudoinverse is a vector,
%we can simplify this procedure
elseif c_flag==2
C=(d_F(:,iq(1))'*ER_f(:,iq(1)))/(d_F(:,iq(1))'*d_F(:,iq(1)));

% if d_F is not along ER_f, then C will be small, not much will happen
```

```
% this is the case for slow converging problems.  let's change the
% error used to calculate C, by adding d_F to ER_f by an amount inversely
% proportionally to the dot product of the two.
elseif c_flag==6
('used new C algorithm')
C=(norm(ER_f(:,iq(1)))^2)/...
(dot(d_F(:,iq(1)),ER_f(:,iq(1)) ) );
end %end c_flag if
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%calculate maximum value of C allowed
%(and meet convergence reqt's)
%the calculation below assumes that f(i) lies
%in zone B1 of th contact force zones
num=0;
den=0;
I_count=0;
for i=1:n
for j=1:n
if i~=j & i<j
I_count=I_count+1;
jj=(i-1)*n+j;
ii=(j-1)*n+i;
num=num+w(i,j)*ER_dot(jj)+w(j,i)*ER_dot(ii);
%this is like: w_ij*er_dot_ij+w_ji*er_dot_ji
end
end
den=den+sum(w(i,:))^2;
end
```

```
C_max=num/den*mu3;

%[C C_max]

if C>C_max
('C exceeded C_max and was corrected')
% C=C_max;
end

% update F_i
F_i_old=F_i_old+C*d_F;

% calculate current contact forces
F(:,in)=F_e(:,in)+F_i_old(:,in);

for i=1:n
Fc(:,i)=T_i(3*i-2:3*i,1:3)'*F(:,i);
%convert to local frame
theta(i)=atan2(Fc(3,i),Fc(2,i));
%update angle
    end
numit=numit+1; %increment interation number
if Ap_flag==1  | Fp_flag==1
%record Fc so that the change in contact force may
  %be graphically displayed
for ic=1:n
F_IT(ic*3-2:ic*3,numit)=F(:,ic);
end
end
```

```
% is any current contact force outside of friction cone
%F
for i=1:n
fc_check(i)=-e_n_oi(:,i)'*F(:,i)+eta*norm(F(:,i));
end

end %end while loop for contact stability

if numit>=n_max
numit=n_max;
('Solution does not satisfy constraints')
end
```

B.1.4 mem_edci.m.

```
% subroutine which calculates the membership of a given input vector
% with respect to the triangular fuzzy membership functions defined
% by the vector of 'centers.' Use this version of membership calc-
% ulation for external force, dot product of internal and external
% forces,
% cross product of internal force and friction cone vectors, and output
% weights associated with each internal force.
% See cone_ctr.m

% Mark Hunter, 1 August 95, Air Force Institute of Technology, WPAFB, OH.


function [mf,mu]=mem_edci(x,centers)

%mf is the membership function label of the first non-zero membership
```

```
%mu(i,1) is the value of the membership of x(i) with repect to mf(i)
%index(i) refers to the number of membership functions crossed by x(i)

n=max(size(x)); %number of input data scalars
m=max(size(centers)); %number of membership functions

% initialize the two matrices
mf=zeros(n,2);
mu=mf;

for i=1:n;

mf_inc=max(find(x(i)>=centers));

   if mf_inc==[]
      mf(i,:)=[1 2];
      mu(i,:)=[1 0];

   elseif mf_inc==m
      mf(i,:)=[m-1 m];
      mu(i,:)=[0 1];

   else
      mf(i,:)=[mf_inc mf_inc+1];

mu(i,1)= (centers(mf_inc+1) - x(i))/...
(centers(mf_inc+1) - centers(mf_inc));
mu(i,2)= (x(i) - centers(mf_inc))/...
(centers(mf_inc+1) - centers(mf_inc));
   end
```

```
end
```

B.1.5 a2_plot.m.

```
function a2_plot(F_IT,numit,E_IT)
% this function opens figure windows for each contact and draws
% the contact forces
% as the FLRS algorithm calculates the next iteration
global n mu T_i r Q

% degrees to radians
deg2rad=pi/180;

colors=hsv(80); %set colors equal to matrix of default color map, hsv
bcolor=1; %base contact color

%figure window positions and sizes
Pos=[ .33 .66 .33 .33;
.66 0 .33 .33;
.33 0 .33 .33;
0 0 .33 .33;
0 .4 .33 .33;
.66 .4 .33 .33];

%for display test purposes
%numit=1;

%display object as a unit radius sphere
```

```
f_hdl(1)=figure(1);
clf

set(f_hdl(1),'Units','normal','Position',Pos(1,:),'MenuBar','none'  )
set(gca,'NextPlot','add');

%plot contact surface (lies along y-axis) and friction cone
%plot in x-y plane
ro=1;
for po=1:37
theta=(po-1)*10;
theta=deg2rad*theta;
sfco(1,po)=ro*cos(theta);
sfco(2,po)=ro*sin(theta);
sfco(3,po)=0;
end
so_hdl=plot3(sfco(1,:)', sfco(2,:)', sfco(3,:)','c');
%draw surface line, cyan
set(so_hdl,'LineWidth',1); %set line width to 1

axis([-1 1 -1 1 -1 1])
axis('square');
axis('off');
view(2)
% hold on

%plot in x-z plane
sfco(3,:)=sfco(2,:);
sfco(2,:)=zeros(size(sfco(2,:)));
so_hdl=plot3(sfco(1,:)', sfco(2,:)', sfco(3,:)','c');
```

```
%draw surface line, cyan
set(so_hdl,'LineWidth',1); %set line width to 1

%plot the input wrench, Q
q_hdl=arrow(Q(1:3), [0 0 0]'); %draw wrench yellow
arrow(q_hdl,'Length',10,'LineWidth',2,'Color','y');
for i=1:n
%plot solution contact forces
cfs_hdl=arrow([0  0 0]+r(:,i)',...
[F_IT((i-1)*3+1,numit) F_IT((i-1)*3+2,numit)...
F_IT((i-1)*3+3,numit)]+r(:,i)');
arrow(cfs_hdl,'Length',10,'BaseAngle',45,'LineWidth',2,'Color',...
colors(80,:) );

%plot both halves of friction cone in green
k_fc=.5; %scale friction cone
%plot in x-y plane
fc_n=T_i(3*i-2:3*i,1:3)*[1 -mu 0]'*k_fc;
fc_p=T_i(3*i-2:3*i,1:3)*[1 mu 0]'*k_fc;

fc_hdl=plot3([0 fc_n(1)]'+[r(1,i) r(1,i)]',[0 fc_n(2)]'...
+[r(2,i) r(2,i)]',[0 fc_n(3)]'+[r(3,i) r(3,i)]','g');
fc_hdl=plot3([0 fc_p(1)]'+[r(1,i) r(1,i)]',[0 fc_p(2)]'...
+[r(2,i) r(2,i)]',[0 fc_p(3)]'+[r(3,i) r(3,i)]','g');

%plot in x-z plane
fc_n=T_i(3*i-2:3*i,1:3)*[1 0 -mu]'*k_fc;
fc_p=T_i(3*i-2:3*i,1:3)*[1 0 mu]'*k_fc;

fc_hdl=plot3([0 fc_n(1)]'+[r(1,i) r(1,i)]',[0 fc_n(2)]'...
```

```
+[r(2,i) r(2,i)]',[0 fc_n(3)]'+[r(3,i) r(3,i)]','b');
fc_hdl=plot3([0 fc_p(1)]'+[r(1,i) r(1,i)]',[0 fc_p(2)]'...
+[r(2,i) r(2,i)]',[0 fc_p(3)]'+[r(3,i) r(3,i)]','b');
end
drawnow
%this function plots the contact force convergence history
%for each CFA solution
%numit - number of iterations for convergence
%F_IT - matrix of contact forces applied during each step of soln.
%in object frame, column length = 3*n, row length = numit
%since forces are in local contact frame, only need x and y components
%for display purposes.

if size(E_IT)>1
e_flag=1; %draw in the contact errors
end
for i=1:n
% determine axis scale to show the force changes throughout
% the convergence
% and make the axes square, ie all axes are the same length

xme=(F_IT((i-1)*3+1,1:numit)+E_IT((i-1)*3+1,1:numit));
yme=(F_IT((i-1)*3+2,1:numit)+E_IT((i-1)*3+2,1:numit));
zme=(F_IT((i-1)*3+3,1:numit)+E_IT((i-1)*3+3,1:numit));
xm=F_IT((i-1)*3+1,1:numit);
ym=F_IT((i-1)*3+2,1:numit);
zm=F_IT((i-1)*3+3,1:numit);

xmax=max([max(xm) max(xme) .05]);
xmin=min([min(xm) min(xme) -.05]);
```

```
ymax=max([max(ym) max(yme) .05]);
ymin=min([min(ym) min(yme) -.05]);
zmax=max([max(zm) max(zme) .05]);
zmin=min([min(zm) min(zme) -.05]);

dx=xmax-xmin;
dy=ymax-ymin;
dz=zmax-zmin;

if dx>dy & dx>dz
ymax=ymax+(dx-dy)/2;
ymin=ymin-(dx-dy)/2;
zmax=zmax+(dx-dz)/2;
zmin=zmin-(dx-dz)/2;
elseif dy>dx & dy>dz
xmax=xmax+(dy-dx)/2;
xmin=xmin-(dy-dx)/2;
zmax=zmax+(dy-dz)/2;
zmin=zmin-(dy-dz)/2;
else %dz>dx & dz>dy
xmax=xmax+(dz-dx)/2;
xmin=xmin-(dz-dx)/2;
ymax=ymax+(dz-dy)/2;
ymin=ymin-(dz-dy)/2;
end

a_limit(i,:)=[xmin xmax ymin ymax zmin zmax];
f_hdl(i+1)=figure(i+1);
clf
set(f_hdl(i+1),'Units','normal','Position',Pos(i+1,:),'MenuBar','none' )
```

```
ab_hdl(i)=axes;
set(ab_hdl(i),'NextPlot','add')
axis(a_limit(i,:));
view(2);
xlabel('Xo');
ylabel('Yo');
zlabel('Zo');
axis('square');
hold on;

%plot contact surface (lies along y-axis) and friction cone
%plot in x-y plane
ri=norm(r(:,i));
for pi=1:19
theta=(pi-1)*5+135;
theta=deg2rad*theta;
sfc(1,pi)=ri+ri*cos(theta);
sfc(2,pi)=ri*sin(theta);
sfc(3,pi)=0;
end
csfc=T_i(3*i-2:3*i,1:3)*sfc;
s_hdl=plot3(csfc(1,:)', csfc(2,:)', csfc(3,:)','c');
%draw surface line, cyan
set(s_hdl,'LineWidth',2); %set line width to 2
%plot in x-z plane
sfc(3,:)=sfc(2,:);
sfc(2,:)=zeros(size(sfc(2,:)));
csfc=T_i(3*i-2:3*i,1:3)*sfc;
s_hdl=plot3(csfc(1,:)', csfc(2,:)', csfc(3,:)','c');
%draw surface line, cyan
```

```
set(s_hdl,'LineWidth',2); %set line width to 2
%plot large white arrows to indicate point contacts
%
pc_start=[-.2 0 0]';
pc_start=T_i(3*i-2:3*i,1:3)*pc_start;
pc_end=[0 0 0]';
pc_hdl=arrow(pc_start',pc_end');
arrow(pc_hdl,'Length',30,'LineWidth',4,'Color','w');

%plot both halves of friction cone in green

%plot in x-y plane
fc_n=T_i(3*i-2:3*i,1:3)*[1 -mu 0]';
fc_p=T_i(3*i-2:3*i,1:3)*[1 mu 0]';

fc_hdl=plot3([0 fc_n(1)]',[0 fc_n(2)]',[0 fc_n(3)]','g');
fc_hdl=plot3([0 fc_p(1)]',[0 fc_p(2)]',[0 fc_p(3)]','g');

%plot in x-z plane
fc_n=T_i(3*i-2:3*i,1:3)*[1 0 -mu]';
fc_p=T_i(3*i-2:3*i,1:3)*[1 0 mu]';

fc_hdl=plot3([0 fc_n(1)]',[0 fc_n(2)]',[0 fc_n(3)]','b');
fc_hdl=plot3([0 fc_p(1)]',[0 fc_p(2)]',[0 fc_p(3)]','b');

drawnow

%plot external contact force
% afo_hdl(i)=axes;
% axis(a_limit(i,:));
```

```
% view(2);
% axis('square');
% axis('off');
fo_hdl=arrow([0  0 0],...
 [F_IT((i-1)*3+1,1) F_IT((i-1)*3+2,1) F_IT((i-1)*3+3,1)]);
arrow(fo_hdl,'BaseAngle',45,'LineWidth',2,'Color',colors(68,:));

end

for j=2:numit
for i=1:n
%plot contact forces

figure(f_hdl(i+1)); %make figure f_hdl(i) the current figure
% af_hdl(i)=axes;
% axis(a_limit(i,:));
% view(2);
% axis('square');
% axis('off');
% hold on;
fcolor=bcolor+3*j;
df_hdl=arrow([F_IT((i-1)*3+1,j-1) F_IT((i-1)*3+2,j-1)...
  F_IT((i-1)*3+3,j-1)],...
[F_IT((i-1)*3+1,j) F_IT((i-1)*3+2,j)...
  F_IT((i-1)*3+3,j)]);
arrow(df_hdl,'BaseAngle',45,'LineWidth',2,'Color',colors(fcolor,:));

if e_flag==1 & max(abs(E_IT((i-1)*3+1:(i-1)*3+3,j-1)))>0
%plot contact force errors
e_hdl=arrow([F_IT((i-1)*3+1,j-1)...
```

```
  F_IT((i-1)*3+2,j-1) F_IT((i-1)*3+3,j-1)],...
[(F_IT((i-1)*3+1,j-1)+E_IT((i-1)*3+1,j-1)),...
(F_IT((i-1)*3+2,j-1)+E_IT((i-1)*3+2,j-1)),...
(F_IT((i-1)*3+3,j-1)+E_IT((i-1)*3+3,j-1))]);
arrow(e_hdl,'BaseAngle',16,'Color','w');
end
%drawnow
end
end
```

B.1.6 fp_plot.

```
function fp_plot(F_IT,numit)

global n

%for display test purposes
%numit=1;

% degrees to radians
deg2rad=pi/180;

%this function plots the contact force convergence history
%for each CFA solution
%numit - number of iterations for convergence
%F_IT - matrix of contact forces applied during each step of soln.
%in object frame, column length = 3*n, row length = numit
%since forces are in local contact frame, only need x and y components
%for display purposes.
```

```
i=1;
figure

axis([1 numit min([min(F_IT((i-1)*3+1,:))
min(F_IT((i-1)*3+2,:))])-.5 ...
max([max(F_IT((i-1)*3+1,:)) ...
max(F_IT((i-1)*3+2,:))])+.5 ])
hold on
p_hdl= plot((1:numit),F_IT((i-1)*3+1,1:numit),'ro')
set(p_hdl,'LineWidth',2); %set line width to 2
p_hdl= plot((1:numit),F_IT((i-1)*3+2,1:numit),'g*')
set(p_hdl,'LineWidth',2); %set line width to 2
p_hdl= plot([1 numit],[-1.25 -1.25],'w')
set(p_hdl,'LineWidth',2); %set line width to 2
p_hdl= plot([1 numit],[.5 .5],'w')
set(p_hdl,'LineWidth',2); %set line width to 2

title('FLRS Two Contact Solution')
xlabel('Iteration No.');
ylabel('Contact Force');
axis('square');

legend('ro','F1x','g*','F1y')
i=2;
figure
axis([1 numit min([min(F_IT((i-1)*3+1,:)) min(F_IT((i-1)*3+2,:))])-.5 ...
max([max(F_IT((i-1)*3+1,:)) max(F_IT((i-1)*3+2,:))])+.5 ])
hold on
p_hdl= plot((1:numit),F_IT((i-1)*3+1,1:numit),'ro')
set(p_hdl,'LineWidth',2); %set line width to 2
```

```
p_hdl= plot((1:numit),F_IT((i-1)*3+2,1:numit),'g*')
set(p_hdl,'LineWidth',2); %set line width to 2
p_hdl= plot([1 numit],[2.25 2.25],'w')
set(p_hdl,'LineWidth',2); %set line width to 2
p_hdl= plot([1 numit],[.5 .5],'w')
set(p_hdl,'LineWidth',2); %set line width to 2


title('FLRS Two Contact Solution')
xlabel('Iteration No.');
ylabel('Contact Force');
axis('square');
legend('ro','F2x','g*','F2y')
```

Appendix C. Real-Time Code

The following C code is the FLRS algorithm incorporated into Chimera compliant modules. The module *cfaload.c* generates the commanded object wrenches and records the time elapsed for a solution. The module *cfa.c* actually solves for the contact forces given the object wrench and grasp configuration data. Functions not included include standard matrix manipulation routines by H.T. Lau, *Numerical Library in C for Scientists and Engineers*, CRC Press, Inc., 1995, ISBN 0-8493-7376-X.

C.1 cfaload.c

```
/* ********************************************************************** */
/*                                                                        */
/* cfaload.c                                                              */
/* */
/*       created by Mark W Hunter          12-13-95          */
/*       Air Force Institute of Technology          */
/* */
/*       modified: */
/* */
/*          $ date $ $ initials $ $ comments $          */
/* */
/*       reviewed by $ Some name here $ $ date $ */
/* */
/* ---------------------------------------------------------------------- */
/* */
/* cfaload: this module is used to test the cfa module by  */
/* sending various wrench commands and sending the cfa contact */
/* dforce results to file along with cpu time required for  */
/* solution. */
```

```
/* */
/* State variable table: */
/* INCONST: none                                    */
/* OUTCOST: N_CONTACT - number of contacts   */
/* MU -  coeffecient of friction between */
/* manipulators and object surface */
/* */
/* INVAR: S_FLAG -    signal flag, determines  */
/*     which module should be  */
/*     currently running */
/* F_CNT -     contact force output    */
/*     from "cfa" */
/* */
/* OUTVAR: WRENCH -    commanded object wrench */
/* CNT_POS -   contact positions  */
/*     in object frame */
/* X_CNT -     orientation of contact x-axis*/
/*     (which is an inward pointing*/
/*     normal to surface) in object*/
/*     frame. */
/* Y_CNT -     orientation of contact y-axis*/
/*     (tangent to contact surface)*/
/* S_FLAG -    signal flag, determines  */
/*     which module should be  */
/*     currently running */
/* */
/*               FC_OFFSET - friction cone offset for grasp stability  */
/*               CONV_OFFSET - solution convergence offset(.1 nominal) */
/*                */
/* */
```

```
/* Special notes: */
/*                                                    */
/* */
/* ********************************************************************** */


/* ********************************************************************** */
/* include files */
/* ********************************************************************** */

#include <chimera.h>
#include <sbs.h>
#include <string.h>    /* string functions */
#include <stdio.h>     /* standard input/output library */
#include <stdarg.h>
#include <ctype.h>
#include <stdlib.h> /* standard library */
#include <math.h> /* standard math library */
/*#include <cfa.h>*/

/* ********************************************************************** */
/* macro definitions */
/* ********************************************************************** */

#define PI 3.141592654
float *temp;

/* ********************************************************************** */
/* module 'Local_t' definition as required by Chimera */
/* ********************************************************************** */
```

```
typedef struct {
  char fname[80];
  char title[80];
  int *n;
  float *mu;
  float *del_X;
  float *fc_x;
  float *Q; /*Q[7]*/
  int iQ_ctr;
  float *R_temp; /*R_temp[16]*/
  float *XC_temp;/*XC_temp[16]*/
  float *YC_temp;/*YC_temp[16]*/
  float *F_temp;
  int *s_flag;
  svarVar_t *s_flag_;
  float *time;
  } cfaloadLocal_t;

FILE *rec;


/* ********************************************************************** */
/* module initialization as required by Chimera */
/* ********************************************************************** */

SBS_MODULE(cfaload);


/* ********************************************************************** */
```

```
/* functions */
/* ********************************************************************** */
 /* function declarations */


/* ********************************************************************** */
/* cfaloadInit Initialize the module. */
/* ********************************************************************** */

int cfaloadInit(cinfo, local, stask)
cfigInfo_t *cinfo;
cfaloadLocal_t *local;
sbsTask_t *stask;
{
   sbsSvar_t *svar = &stask->svar;


/* Get pointers to state variables (CONST). */


    local->n = svarTranslateValue(svar->vartable, "N_CONTACT", int);
    local->mu = svarTranslateValue(svar->vartable, "MU", float);

/* Get pointers to state variables. */

        local->s_flag_          = svarTranslate(svar->vartable, "S_FLAG");
        local->s_flag           = svarValue(local->s_flag_, int);

local->F_temp = svarTranslateValue(svar->vartable, "F_CNT", float);
local->time = svarTranslateValue(svar->vartable, "CPU_TIME", float);
```

```
    local->Q = svarTranslateValue(svar->vartable, "WRENCH", float);
    local->R_temp = svarTranslateValue(svar->vartable, "CNT_POS", float);
    local->XC_temp = svarTranslateValue(svar->vartable, "X_CNT", float);
    local->YC_temp = svarTranslateValue(svar->vartable, "Y_CNT", float);
       local->fc_x = svarTranslateValue(svar->vartable, "FC_OFFSET", float);
local->del_X = svarTranslateValue(svar->vartable, "CONV_OFFSET", float);


/* define OUTCONST contact parameters */

local->n[0]=3; /* number of contacts */
local->mu[0]=.5; /* coefficient of friction*/


  return (int) local;
}


/* ********************************************************************** */
/* cfaloadReinit Re-Initialize the module. */
/* ********************************************************************** */

int cfaloadReinit(local, stask)
cfaloadLocal_t *local;
sbsTask_t *stask;
{
  return I_OK;
}
```

```
/* ********************************************************************** */
/* cfaloadOn Start up the module. */
/* ********************************************************************** */

int cfaloadOn(local, stask)
cfaloadLocal_t *local;
sbsTask_t *stask;
{

kprintf("cfaload: ON\n");
   printf("Starting the Contact Force Assignment Test Module \n");
/* initialize signal flag */


/* explicit write to state variable table  */

local->s_flag[0]=2;
svarWrite(local->s_flag_);

/* define contact parameters */
local->del_X[0]=.1; /* convergence buffer */
local->fc_x[0]=0; /* stability buffer */

/*  define contact postions */
local->R_temp[0]=0;

local->R_temp[1]=1;
local->R_temp[2]=0;
```

```
local->R_temp[3]=0;

local->R_temp[4]=0;
local->R_temp[5]=-1;
local->R_temp[6]=0;

local->R_temp[7]=-1;
local->R_temp[8]=0;
local->R_temp[9]=0;


/*  define local x-axis in object frame */
local->XC_temp[0]=0;

local->XC_temp[1]=-1;
local->XC_temp[2]=0;
local->XC_temp[3]=0;

local->XC_temp[4]=0;
local->XC_temp[5]=1;
local->XC_temp[6]=0;

local->XC_temp[7]=1;
local->XC_temp[8]=0;
local->XC_temp[9]=0;


/*  define local y-axis in object frame */
local->YC_temp[0]=0;
```

```
local->YC_temp[1]=0.0;
local->YC_temp[2]=-1.0;
local->YC_temp[3]=0.0;

local->YC_temp[4]=-1.0;
local->YC_temp[5]=0.0;
local->YC_temp[6]=0.0;

local->YC_temp[7]=0;
local->YC_temp[8]=1;
local->YC_temp[9]=0;

/* initalize wrench values */
local->Q[0]=0;

local->Q[1]=0;
local->Q[2]=0;
local->Q[3]=0;
local->Q[4]=0;
local->Q[5]=0;
local->Q[6]=0;

local->iQ_ctr=0;

kprintf("The current value of s-flag should
be 2, s_flag= %d \n", local->s_flag[0]);

/* open a file to print results of test */

if ((rec = fopen("TEST.DAT","wt")) == NULL)
```

```
{
kprintf("Can not create '%s'\n","TEST.DAT");
}



  return I_OK;
}


/* ********************************************************************** */
/* cfaloadCycle Process module information. */
/* ********************************************************************** */

int cfaloadCycle(local, stask)
cfaloadLocal_t *local;
sbsTask_t *stask;
{

if (local->s_flag[0]==2)   /*if s_flag==2, time to generate Q,*/
/* if s_flag==1, ignore this cycle */
{

fprintf(rec,"%d \t %f
\t %f \t %f \t %f
\t %f \t %f \t %f
\t %f \t %f \t %f
\t %f \t %f \t %f
\t %f \t %f \t %f \n",
  local->iQ_ctr, local->time[0],
```

```
  local->F_temp[1], local->F_temp[2], local->F_temp[3],
  local->F_temp[4], local->F_temp[5], local->F_temp[6],
  local->F_temp[7], local->F_temp[8], local->F_temp[9],
  local->F_temp[10], local->F_temp[11], local->F_temp[12],
  local->F_temp[13], local->F_temp[14], local->F_temp[15]);



kprintf("This is the current value of theta, %d \n", local->iQ_ctr+1);

local->iQ_ctr=local->iQ_ctr+1;
local->Q[1]=cos(local->iQ_ctr*PI/36);
local->Q[2]=sin(local->iQ_ctr*PI/36);



local->s_flag[0]=1; /* update S_FLAG to 1, so that */
/* cfa will execute next cycle */

kprintf( "The current value of s_flag should be 1,
 s_flag= %d \n",local->s_flag[0]);

if (local->iQ_ctr>72)/* finished with the input set, end test */
{
local->s_flag[0]=3;
kprintf( "The input set is now finished, kill this process \n");
}


}
```

```
  return I_OK;
}


/* ********************************************************************** */
/* cfaloadOff Stop the module. */
/* ********************************************************************** */

int cfaloadOff(local, stask)
cfaloadLocal_t *local;
sbsTask_t *stask;
{

  kprintf("cfaload: OFF\n");
  printf("Stopping the Contact Force Assignment Test Module \n");

fclose(rec);

  return I_OK;
}


/* ********************************************************************** */
/* cfaloadKill Clean up after the module. */
/* ********************************************************************** */

int cfaloadKill(local, stask)
cfaloadLocal_t *local;
```

```
sbsTask_t *stask;
{
  kprintf("cfaload: FINISHED\n");
  return I_OK;
}


/* ********************************************************************** */
/* cfaloadError Attempt automatic error recovery. */
/* ********************************************************************** */

int cfaloadError(local, stask, mptr, errmsg, errcode)
cfaloadLocal_t *local;
sbsTask_t *stask;
errModule_t *mptr;
char *errmsg;
int errcode;
{
  /* Return after not correcting error. */

  return SBS_ERROR;
}


/* ********************************************************************** */
/* cfaloadClear Clear error state of the module. */
/* ********************************************************************** */

int cfaloadClear(local, stask, mptr, errmsg, errcode)
cfaloadLocal_t *local;
```

```
sbsTask_t *stask;
errModule_t *mptr;
char *errmsg;
int errcode;
{
  /* Return after not clearing error. */

  sbsNewError(stask, "Clear not defined, still in error state", errcode);
  return SBS_ERROR;
}


/* ********************************************************************** */
/* cfaloadSet Set module parameters. */
/* ********************************************************************** */

int cfaloadSet(local, stask)
cfaloadLocal_t *local;
sbsTask_t *stask;
{
  return I_OK;
}


/* ********************************************************************** */
/* cfaloadGet Get module parameters. */
/* ********************************************************************** */

int cfaloadGet(local, stask)
cfaloadLocal_t *local;
```

```
sbsTask_t *stask;
{
  return I_OK;
}


/* ********************************************************************** */
/* cfaloadSync For modules which synchronize to*/
/* something other than the clock. */
/* ********************************************************************** */

int cfaloadSync(local, stask)
cfaloadLocal_t *local;
sbsTask_t *stask;
{
  return I_OK;
}
```

C.2 cfa.c

```
/* ********************************************************************** */
/*                                                    */
/* cfa.c                                             */
/* */
/*       created by Mark W Hunter          12-05-95         */
/*       Air Force Institute of Technology         */
/* */
/*       modified: */
/* */
```

```
/*          $ date $ $ initials $ $ comments $          */
/* */
/*       reviewed by $ Some name here $ $ date $ */
/* */
/* ------------------------------------------------------------------ */
/* */
/* cfaload: this module is used to test the cfa module by  */
/* sending various wrench commands and sending the cfa contact */
/* dforce results to file along with cpu time required for  */
/* solution. */
/* */
/* State variable table: */
/* INCONST: N_CONTACT - number of contacts   */
/* MU -  coeffecient of friction between */
/* manipulators and object surface */
/* OUTCONST: none                                    */
/* */
/* INVAR: WRENCH -    commanded object wrench */
/* CNT_POS -   contact positions  */
/*     in object frame */
/* X_CNT -     orientation of contact x-axis*/
/*     (which is an inward pointing*/
/*     normal to surface) in object*/
/*     frame. */
/* Y_CNT -     orientation of contact y-axis*/
/*     (tangent to contact surface)*/
/* S_FLAG -    signal flag, determines  */
/*     which module should be  */
/*     currently running */
/* */
```

```
/*                FC_OFFSET - friction cone offset for grasp stability  */
/*                CONV_OFFSET - solution convergence offset(.1 nominal) */
/*                 */
/* OUTVAR: F_CNT -     contact forces necessary to */
/*     generate the required object*/
/*     wrench                        */
/* S_FLAG -    signal flag, determines  */
/*     which module should be  */
/*     currently running */
/* */
/* Special notes: */
/*                                          */
/* */
/* ********************************************************************** */


/* ********************************************************************** */
/* include files */
/* ********************************************************************** */

#include <chimera.h>
#include <sbs.h>
#include <string.h>    /* string functions */
#include <stdio.h>     /* standard input/output library */
#include <stdarg.h>
#include <ctype.h>
#include <stdlib.h> /* standard library */
#include <math.h> /* standard math library */
/*#include <cfa.h>*/
```

```
/* ********************************************************************** */
/* macro definitions */
/* ********************************************************************** */

#define PI 3.141592654

/* ********************************************************************** */
/* module 'Local_t' definition as required by Chimera */
/* ********************************************************************** */

typedef struct {
char fname[80]; /* file name for output data                    */
    char title[80]; /* title of output vector or matrix          */
       int *n;         /* number of contacts (assume 5 is the max number) */
int i3n,inn,num_I;

    float *mu;       /* coeffecient of friction */
    float *del_X; /* convergence buffer for fuzzy solver */
float *fc_x;  /* friction cone offset, for contact stability */

/* rulebase for fuzzy logic system 7x7 plus zero'th row and col offsets */
int **RB_ef;

/* fuzzy domain for dot product information 7x1 plus zero'th element*/
float *c_dot;

    /* fuzzy range of e_I effectiveness 7x1 plus zero'th element */
float *cfd_wt;

       /* area of fuzzy membership functions of cfd_wt 7x1 */
```

```
float *afd_wt;

/* number of membership functions which comprise the domain of inputs */
/* to the fuzzy analysis portion of this code */
int nc;

/* input contact positions as col. vectors, allow up to five contacts */
float **r;

/* xc[][] and yc[][] are inputs of contact orientation */
       /* *c is the contact *-axis direction in object frame */
       float **xc, **yc, **zc;

       float **Fo; /* external contact force solution, 3xn */

       float *Q; /* object wrench 6x1 plus zero'th element */

       float **F; /* matrix of contact forces 3xn */

       float **T_i; /*[3+1][d3n1];/* transformation matrix  3x3n */

       float **e_I;/*[3+1][dnn1];/* internal force unit vectors, 3xn*n */

       float **e_Ic;/*[3+1][dnn1];/*e_I in local coordinates for each contact */

       float **W;  /*[6+1][d3n1];/* grasp matrix */
       float **Wp;  /*[6+1][d3n1];/* psuedoinverse of grasp matrix */


/* below are arrays used in 'h_sol'  */
```

```
    float **F_i_old, **Fc, *theta, *fc_check, **ER_f, **ER_fc;
float *vl_1;
    float *ER_dot, *wt_tot;
float **mu_ERd, **wt_ef, *nER_f, **d_F, **d_Fc;
float *d;
int **imf_ERd;
/* these variables are used for temporary storage of state varibles  which */
/* are in vector form and must be converted to matrix form */
float *F_temp, *R_temp, *XC_temp, *YC_temp;
float *time;
int *s_flag;

  } cfaLocal_t;

/* ********************************************************************** */
/* module initialization as required by Chimera */
/* ********************************************************************** */

SBS_MODULE(cfa);


/* ********************************************************************** */
/* functions */
/* ********************************************************************** */
   /* function declarations */
   /* unless otherwise noted, these functions are authored by: */
   /* H.T. Lau */
   /* Numerical Library in C for Scientists and Engineers */
   /* 1995 CRC Press, Inc. */
   /* ISBN 0-8493-7376-X */
```

```
   void inivec(int, int, float [], float);
/* initializes a real vector to a given value */
   void inimat(int, int, int, int, float **, float);
/* initializes a real matrix */
   void ini_imat(int, int, int, int, int **, int);
/* initializes an int matri */
   float matvec(int, int, int, float **, float []);
/* s=mat(row)*vec */
   float tamvec(int, int, int, float **, float []);
/* s=mat(col)*vec */
   float matmat(int, int, int, int, float **, float **);
   void fulmatvec(int, int, int, int, float **, float [], float []);
/* c[]=mat*vec */
   void fultamvec(int, int, int, int, float **, float [], float []);
/* c[]=mat(col)*vec */
   void dupmat(int, int, int, int, float **, float **);
/* copy b[][] into a[][] */
   float vecvec(int, int, int, float [], float []);
/* inner vector product */


   float **allocate_real_matrix(int, int, int, int);
   int **allocate_int_matrix(int, int, int, int);
   float *allocate_real_vector(int, int);
   void free_real_matrix(float **, int, int, int);
   void free_int_matrix(int **, int, int, int);
   void free_real_vector(float *, int);

   int psdinv(float **, int, int, float []);/* psuedo inverse solution */
```

```
   int qrisngvaldec(float **, int, int, float [], float **, float []);
void hshreabid(float **, int, int, float [], float [], float []);
float tammat(int, int, int, int, float **, float **);
float mattam(int, int, int, int, float **, float **);
void elmcol(int, int, int, int, float **, float **, float);
void elmrow(int, int, int, int, float **, float **, float);
void psttfmmat(float **, int, float **, float []);
          /* matmat, elmcol */
   void pretfmmat(float **, int, int, float []);
          /* tammat, elmcol */
   int qrisngvaldecbid(float [], float [], int, int, float **, float **,
 float []);
   void rotcol(int, int, int, int, float **, float, float);
   void psdinvsvd(float **, float [], float **, int, int, float []);
   /*matvec*/

   void system_error(char []);

   /* euclidean norm of vector (mhunter) */
   float euclnorm(int, int, float []);

   /* cross product of two spatial (x,y,z) input vectors (mhunter) */
   void cross(int, int, float[], float[], float[]);

   /* print out real matrix to file (mhunter) */
   void m_out(float **, int, int, int, int, char[], char[]);

   /* print out int matrix to file (mhunter) */
   void im_out(int **, int, int, int, int, char[], char[]);
```

```
   /* print out real vector to file (mhunter) */
   void v_out(float [], int, int, char[], char[]);

   /* print out real scalar to file (mhunter) */
   void s_out(float, char[], char[]);


   /* solve for the contact forces given the object wrench (mhunter) */
   void h_sol( float, float, float, int, int, int,
int **, float [], int, float [], float [],
float **, float **, float **,
float **, float **, float [], float **,
float **, float **, float [],
float [], float **, float **, float [],
float [], float [], int **, float **, float **,
float [], float **, float **, float []);

   /* determine maximum value from given vector (mhunter) */
   float mymax(int, int, float []);

   /* determine maximum value from given vector (mhunter) */
   float mymin(int, int, float []);

   /* fuzzy inference mechanism (mhunter) */
   void mem_edci( int **, float **, float [], int, float [], int);


/* ********************************************************************** */
/* cfaInit Initialize the module. */
/* ********************************************************************** */
```

```
int cfaInit(cinfo, local, stask)
cfigInfo_t *cinfo;
cfaLocal_t *local;
sbsTask_t *stask;
{
   sbsSvar_t *svar = &stask->svar;

/* Get pointers to state variables (CONST). */

    local->n = svarTranslateValue(svar->vartable, "N_CONTACT", int);
    local->mu = svarTranslateValue(svar->vartable, "MU", float);
local->s_flag = svarTranslateValue(svar->vartable, "S_FLAG", int);

    local->Q = svarTranslateValue(svar->vartable, "WRENCH", float);
    local->R_temp = svarTranslateValue(svar->vartable, "CNT_POS", float);
    local->XC_temp = svarTranslateValue(svar->vartable, "X_CNT", float);
    local->YC_temp = svarTranslateValue(svar->vartable, "Y_CNT", float);
    local->fc_x = svarTranslateValue(svar->vartable, "FC_OFFSET", float);
local->del_X = svarTranslateValue(svar->vartable, "CONV_OFFSET", float);
    local->F_temp = svarTranslateValue(svar->vartable, "F_CNT", float);

local->time = svarTranslateValue(svar->vartable, "CPU_TIME", float);


  return (int) local;
}

/* ********************************************************************** */
/* cfaReinit Re-Initialize the module. */
```

```
/* ********************************************************************** */

int cfaReinit(local, stask)
cfaLocal_t *local;
sbsTask_t *stask;
{
  return I_OK;
}

/* ********************************************************************** */
/* cfaOn Start up the module. */
/* ********************************************************************** */

int cfaOn(local, stask)
cfaLocal_t *local;
sbsTask_t *stask;
{

float v1[4]= {0,0,0,0};
float v2[4]= {0,0,0,0};
float v3[4]= {0,0,0,0};
float r_ij[4]= {0,0,0,0};
float rij[4]= {0,0,0,0};
float r_norm[4]={0,0,0,0};

float I3[4][4]= { {0, 0, 0, 0},
{0, 1, 0, 0},
{0, 0, 1, 0},
{0, 0, 0, 1} };
```

```
float Pi[4][4]={ {0,0,0,0},
{0,0,0,0},
{0,0,0,0},
{0,0,0,0} };

/*  em[] controls exit convergence criteria on psuedoinverse solution */
float em[8]={1.0e-14, 0.0, 1.0e-12, 0.0, 40.0, 0.0, 1.0e-10, 0.0};

int i,j,jj,k;          /* counters */


kprintf("cfa: ON\n");
   kprintf("Starting the Contact Force Assignment Module \n");

local->nc=7;/* number of fuzzy membership functions across input domains */

/* Get pointers to state variables. */

/* local->Q=allocate_real_vector(1,6);

local->R_temp=allocate_real_vector(1,15);
local->XC_temp=allocate_real_vector(1,15);
local->YC_temp=allocate_real_vector(1,15);
local->F_temp=allocate_real_vector(1,15);  */

/* Allocate memory and intialize variables */
local->i3n=3.0 * (local->n[0]);
local->inn=local->n[0] * (local->n[0]);
local->num_I=(local->n[0] * (local->n[0]) - local->n[0])/2;
/* number of internal forces */
```

```
local->RB_ef=allocate_int_matrix(1,7,1,7);
local->c_dot=allocate_real_vector(1,7);
local->cfd_wt=allocate_real_vector(1,7);
local->afd_wt=allocate_real_vector(1,7);
local->r=allocate_real_matrix(1,3,1,local->n[0]);
local->xc=allocate_real_matrix(1,3,1,local->n[0]);
local->yc=allocate_real_matrix(1,3,1,local->n[0]);
local->zc=allocate_real_matrix(1,3,1,local->n[0]);
local->Fo=allocate_real_matrix(1,3,1,local->n[0]);
local->F=allocate_real_matrix(1,3,1,local->n[0]);
local->T_i=allocate_real_matrix(1,3,1,local->i3n);
local->e_I=allocate_real_matrix(1,3,1,local->inn);
local->e_Ic=allocate_real_matrix(1,3,1,local->inn);
local->W=allocate_real_matrix(1,6,1,local->i3n);
local->Wp=allocate_real_matrix(1,local->i3n,1,6);


local->F_i_old=allocate_real_matrix(1,3,1,local->n[0]);
local->Fc=allocate_real_matrix(1,3,1,local->n[0]);
local->theta=allocate_real_vector(1, local->n[0]);
local->fc_check=allocate_real_vector(1, local->n[0]);
local->ER_f=allocate_real_matrix(1,3,1,local->n[0]);
local->ER_fc=allocate_real_matrix(1,3,1,local->n[0]);
local->vl_1=allocate_real_vector(1, 3);
local->ER_dot=allocate_real_vector(1, local->inn);
local->wt_tot=allocate_real_vector(1, local->inn);
local->nER_f=allocate_real_vector(1, local->n[0]);
local->imf_ERd=allocate_int_matrix(1,local->inn,1,2);
local->mu_ERd=allocate_real_matrix(1,local->inn,1,2);
```

```
local->wt_ef=allocate_real_matrix(1,local->nc, 1, local->num_I);
local->d_F=allocate_real_matrix(1,3,1,local->n[0]);
local->d_Fc=allocate_real_matrix(1,3,1,local->n[0]);
local->d=allocate_real_vector(1, local->inn);

/* Reformat *_temp vectors is matrix format as noted below in the example */

for(i=1;i<=local->n[0];i++)
{
for(j=1;j<=3;j++)
{
local->r[j][i]=local->R_temp[(i-1)* (local->n[0])+j];
local->xc[j][i]=local->XC_temp[(i-1)* (local->n[0])+j];
local->yc[j][i]=local->YC_temp[(i-1)* (local->n[0])+j];
}
}

/* example of what input matrices  should look like */
/* r[1][1]=1;  r[1][2]= 0;  r[1][3]=-1; r[1][4]=0;  r[1][5]=0; */
/* r[2][1]=0;  r[2][2]=-1;  r[2][3]=0;  r[2][4]=0;  r[2][5]=0; */
/* r[3][1]=0;  r[3][2]= 0;  r[3][3]=0;  r[3][4]=0;  r[3][5]=0; */
/* */
/* xc[1][1]=-1; xc[1][2]= 0;  xc[1][3]=1;  xc[1][4]=0;  xc[1][5]=0;*/
/* xc[2][1]=0;  xc[2][2]=1;   xc[2][3]=0;  xc[2][4]=0;  xc[2][5]=0;*/
/* xc[3][1]=0;  xc[3][2]= 0;  xc[3][3]=0;  xc[3][4]=0;  xc[3][5]=0;*/
/*  */
/* yc[1][1]=0;  yc[1][2]=-1;  yc[1][3]=0;  yc[1][4]=0;  yc[1][5]=0;*/
/* yc[2][1]=-1; yc[2][2]=0;   yc[2][3]=1;  yc[2][4]=0;  yc[2][5]=0;*/
/* yc[3][1]=0;  yc[3][2]= 0;  yc[3][3]=0;  yc[3][4]=0;  yc[3][5]=0;*/
```

```
/* calculate zc using cross product of xc and yc */

for(j=1; j<=local->n[0]; j++)
{
for(i=1; i<=local->n[0]; i++)
        {
v1[i]=local->xc[i][j];
v2[i]=local->yc[i][j];
        }
   cross(1, 3, v1, v2, v3);

for(i=1;i<=3;i++)
        {
local->zc[i][j]=v3[i];
        }
}

for(i=1; i<=local->n[0]; i++) /* form coordinate transformation matrices */
{
for(k=1;k<=3;k++)
        {
local->T_i[k][3*(i-1)+1]=local->xc[k][i];
local->T_i[k][3*(i-1)+2]=local->yc[k][i];
local->T_i[k][3*(i-1)+3]=local->zc[k][i];
        }
}

for(i=1; i<=local->n[0]; i++) /* form internal force vectors */
{
for(j=1; j<=local->n[0]; j++)
```

```
{
jj=(i-1)*local->n[0]+j;

if(i==j)
{
for(k=1;k<=3;k++)
{
local->e_I[k][jj]=0;
}
}
else
{
          for(k=1;k<=3;k++)
{
r_ij[k]=local->r[k][i]-local->r[k][j];
rij[k]=local->r[k][i]-local->r[k][j];
}

r_norm[jj]=euclnorm(1,3,rij);

for(k=1;k<=3;k++)
{
local->e_I[k][jj]=-r_ij[k]/r_norm[jj];
}
}
}
}

/* in order to calculate e_Ic, which is the internal */
/*force vectors in the */
```

```
/*frame local to themselves, vectors v1 and v2, and */
/*matrix zc will be over written */

for(i=1; i<=local->n[0]; i++)
{
for(j=1; j<=local->n[0]; j++)
{
jj=(i-1)*local->n[0]+j;

          for(k=1;k<=3;k++)
          {
          v1[k]=local->e_I[k][jj];
local->zc[k][1]= local->T_i[k][3*(i-1)+1];
local->zc[k][2]= local->T_i[k][3*(i-1)+2];
local->zc[k][3]= local->T_i[k][3*(i-1)+3];
}

fultamvec(1,3,1,3,local->zc,v1,v2);
/* the returned vector v2, is the result  */
/* of transpose of transformation matrix  */
/* multiplied by the global e_I vectors */
/* associated with each contact.  */
for(k=1;k<=3;k++)
{
local->e_Ic[k][jj]=v2[k];
}
}

} /* end internal force vector formulation */
```

```
/* begin grasp matrix formulation */

for(i=1;i<=local->n[0];i++)
{
Pi[1][2]=-local->r[3][i];
Pi[1][3]= local->r[2][i];
Pi[2][1]= local->r[3][i];
Pi[2][3]=-local->r[1][i];
Pi[3][1]=-local->r[2][i];
Pi[3][2]= local->r[1][i];


for(j=1;j<=3;j++)
{
for(k=1;k<=3;k++)
{
local->W[k][(i-1)*3+j]=I3[k][j];
}

for(k=4;k<=6;k++)
{
local->W[k][(i-1)*3+j]=Pi[k-3][j];
}
}
} /* end grasp matrix formulation */

for(i=1;i<=local->i3n;i++)
{
for(j=1;j<=6;j++)
```

```
{
local->Wp[i][j]=local->W[j][i];
}
    }

i=psdinv(local->Wp,local->i3n,6,em);


/* initialize fuzzy rulebase and domain definitions */


local->RB_ef[1][1]=1; local->RB_ef[1][2]= 1;
local->RB_ef[1][3]=1; local->RB_ef[1][4]=2;
local->RB_ef[1][5]=2; local->RB_ef[1][6]=3;
local->RB_ef[1][7]=4;
local->RB_ef[2][1]=1; local->RB_ef[2][2]= 2;
local->RB_ef[2][3]=2; local->RB_ef[2][4]=3;
local->RB_ef[2][5]=3; local->RB_ef[2][6]=4;
local->RB_ef[2][7]=5;
local->RB_ef[3][1]=1; local->RB_ef[3][2]= 2;
local->RB_ef[3][3]=4; local->RB_ef[3][4]=4;
local->RB_ef[3][5]=4; local->RB_ef[3][6]=5;
local->RB_ef[3][7]=6;
local->RB_ef[4][1]=2; local->RB_ef[4][2]= 3;
local->RB_ef[4][3]=4; local->RB_ef[4][4]=4;
local->RB_ef[4][5]=4; local->RB_ef[4][6]=5;
local->RB_ef[4][7]=6;
local->RB_ef[5][1]=2; local->RB_ef[5][2]= 3;
local->RB_ef[5][3]=4; local->RB_ef[5][4]=4;
local->RB_ef[5][5]=4; local->RB_ef[5][6]=6;
```

```
local->RB_ef[5][7]=7;
local->RB_ef[6][1]=3; local->RB_ef[6][2]= 4;
local->RB_ef[6][3]=5; local->RB_ef[6][4]=5;
local->RB_ef[6][5]=6; local->RB_ef[6][6]=6;
local->RB_ef[6][7]=7;
local->RB_ef[7][1]=4; local->RB_ef[7][2]= 5;
local->RB_ef[7][3]=6; local->RB_ef[7][4]=6;
local->RB_ef[7][5]=7; local->RB_ef[7][6]=7;
local->RB_ef[7][7]=7;

/* fuzzy domain for dot product information 7x1 plus zero'th element*/
local->c_dot[1]=-1;
local->c_dot[2]=-.6;
local->c_dot[3]=-.2;
local->c_dot[4]=0;
local->c_dot[5]=.2;
local->c_dot[6]=.6;
local->c_dot[7]=1;


/* fuzzy range of e_I effectiveness 7x1 plus zero'th element */
local->cfd_wt[1]=-1;
local->cfd_wt[2]=-.5;
local->cfd_wt[3]=-.2;
local->cfd_wt[4]=0;
local->cfd_wt[5]=.2;
local->cfd_wt[6]=.5;
local->cfd_wt[7]=1;

for(i=1;i<=local->nc;i++)
```

```
/* calculate area of membership functions along cfd_wt */
{
if(i==1) local->afd_wt[i]=fabs(local->cfd_wt[i+1]-local->cfd_wt[i])/2;
else if(i==local->nc-1) local->afd_wt[i]=
fabs(local->cfd_wt[i]-local->cfd_wt[i-1])/2;
else  local->afd_wt[i]=(fabs(local->cfd_wt[i]-local->cfd_wt[i-1])+
fabs(local->cfd_wt[i+1]-local->cfd_wt[i]))/2;
}


/* print out various vectors and matrices for checking purposes */

/************************************/
    /* print out variables to file fname */
strcpy(local->fname,"cfa_out");
strcpy(local->title,"Local X-Axes in Object Frame, xc \n");
m_out(local->xc, 1, 3, 1, 3, local->title, local->fname);
/************************************/

/************************************/
    /* print out variables to file fname */
/*strcpy(local->fname,"cfa_out");*/
/*strcpy(local->title,"Local Y-Axes in Object Frame, yc \n");*/
/*m_out(local->yc, 1, 3, 1, 3, local->title, local->fname);*/
/************************************/

/************************************/
    /* print out variables to file fname */
/*strcpy(local->fname,"cfa_out"); */
/*strcpy(local->title,"Local Z-Axes in Object Frame, zc \n");*/
```

```
/*m_out(zc, 1, 3, 1, 3, local->title, local->fname);*/
/* don't bother printing out zc, since this matrix has been recycled */
/***********************************/


/***********************************/
    /* print out variables to file fname */
strcpy(local->fname,"cfa_out");
strcpy(local->title,"Position Vector to Contacts, r \n");
m_out(local->r, 1, 3, 1, 3, local->title, local->fname);
/***********************************/

/***********************************/
    /* print out variables to file fname */
/*strcpy(local->fname,"cfa_out");*/
/*strcpy(local->title,"Transformation Matrix, T_i \n");*/
/*m_out(local->T_i, 1, 3, 1, local->i3n, local->title, local->fname);*/
/***********************************/

/***********************************/
    /* print out variables to file fname */
/*strcpy(local->fname,"cfa_out");*/
/*strcpy(local->title,"Contact Internal Force Vectors, e_I \n");*/
/*m_out(local->e_I, 1, 3, 1, local->inn, local->title, local->fname);*/
/***********************************/

/***********************************/
    /* print out variables to file fname */
/*strcpy(local->fname,"cfa_out");*/
/*strcpy(local->title,"Internal Force Vectors in Contact Frame, e_Ic \n");*/
```

```
/*m_out(local->e_Ic, 1, 3, 1, local->inn, local->title, local->fname);*/
/***********************************/

/***********************************/
    /* print out variables to file fname */
/*strcpy(local->fname,"cfa_out");*/
/*strcpy(local->title,"Grasp Matrix, W \n");*/
/*m_out(local->W, 1, 6, 1, local->i3n, local->title, local->fname);*/
/***********************************/

/***********************************/
    /* print out variables to file fname */
/*strcpy(local->fname,"cfa_out");*/
/*strcpy(local->title,"PsuedoInverse of Grasp Matrix, Wp \n");*/
/*m_out(local->Wp, 1, local->i3n, 1, 6, local->title, local->fname);*/
/***********************************/



  return I_OK;
}


/* ********************************************************************** */
/* cfaCycle Process module information. */
/* ********************************************************************** */

int cfaCycle(local, stask)
cfaLocal_t *local;
sbsTask_t *stask;
```

```
{
static int i,j;

if (local->s_flag[0]==1)   /*if s_flag==1, time to execute h_sol, */
/*if s_flag==2, ignore this cycle */
{

kprintf("About to start the h_sol solution function \n");

local->time[0]=clock();
/* set time to RTPU clock time since system turned on (sec) */


/* for a given object wrench Q, calculate the contact forces */
   h_sol( local->mu[0], local->del_X[0], local->fc_x[0], local->n[0],
local->num_I, local->inn,
local->RB_ef, local->c_dot, local->nc,
local->cfd_wt, local->afd_wt,
local->T_i, local->e_I, local->e_Ic,
local->Fo, local->F, local->Q, local->Wp,
local->F_i_old, local->Fc,
local->theta, local->fc_check, local->ER_f, local->ER_fc,
local->vl_1,local->ER_dot, local->wt_tot, local->imf_ERd,
local->mu_ERd, local->wt_ef, local->nER_f, local->d_F,
local->d_Fc, local->d);


/* Reformat force vectors is matrix format as noted below in the example */

for(i=1;i<=local->n[0];i++)
```

```
{
for(j=1;j<=3;j++)
{
local->F_temp[(i-1)*local->n[0]+j]=local->F[j][i];
}
}
/* set time equal to the time required to execute h_sol */
local->time[0]=clock() - local->time[0];
/* update S_FLAG to 2, so that cfaload will execute next cycle */
  local->s_flag[0]=2;

kprintf("Finished the h_sol solution function \n");


/************************************/
    /* print out variables to file fname */
/*strcpy(local->fname,"cfa_wrench");*/
/*strcpy(local->title,"Input Object Wrench, Q \n");*/
/*v_out(local->Q, 1, 6, local->title, local->fname);*/
/************************************/

/************************************/
    /* print out variables to file fname */
/*strcpy(local->fname,"cfa_out");*/
/*strcpy(local->title,"Contact Forces, Fo \n");*/
/*m_out(local->Fo, 1, 3, 1, local->n[0], local->title, local->fname);*/
/************************************/

   /************************************/
    /* print out variables to file fname */
```

```
/*strcpy(local->fname,"cfa_force");*/
/*strcpy(local->title,"Contact Forces, F \n");*/
/*m_out(local->F, 1, 3, 1, local->n[0], local->title, local->fname);*/
/***********************************/

/***********************************/
    /* print out variables to file fname */
/*strcpy(local->fname,"cfa_time");*/
/*strcpy(local->title," \n");*/
/*s_out(local->time[0], local->title, local->fname);*/
/***********************************/

}  /* end if s_flag */



  return I_OK;
}


/* ********************************************************************** */
/* cfaOff Stop the module. */
/* ********************************************************************** */

int cfaOff(local, stask)
cfaLocal_t *local;
sbsTask_t *stask;
{

/*release memory allocated above */
```

```
free_real_vector(local->R_temp,1);
free_real_vector(local->XC_temp,1);
free_real_vector(local->YC_temp,1);
free_real_vector(local->F_temp,1);
free_int_matrix(local->RB_ef,1,7,1);
free_real_vector(local->c_dot,1);
free_real_vector(local->cfd_wt,1);
free_real_vector(local->afd_wt,1);
free_real_matrix(local->r,1,3,1);
free_real_matrix(local->xc,1,3,1);
free_real_matrix(local->yc,1,3,1);
free_real_matrix(local->zc,1,3,1);
free_real_matrix(local->Fo,1,3,1);
free_real_vector(local->Q,1);
free_real_matrix(local->F,1,3,1);
free_real_matrix(local->T_i,1,3,1);
free_real_matrix(local->e_I,1,3,1);
free_real_matrix(local->e_Ic,1,3,1);
free_real_matrix(local->W,1,6,1);
free_real_matrix(local->Wp,1,local->i3n,1);


free_real_matrix(local->F_i_old,1,3,1);
free_real_matrix(local->Fc,1,3,1);
free_real_vector(local->theta,1);
free_real_vector(local->fc_check,1);
free_real_matrix(local->ER_f,1,3,1);
free_real_matrix(local->ER_fc,1,3,1);
free_real_vector(local->vl_1,1);
free_real_vector(local->ER_dot,1);
```

```
free_real_vector(local->wt_tot,1);
free_real_vector(local->nER_f,1);
free_int_matrix(local->imf_ERd,1,local->inn,1);
free_real_matrix(local->mu_ERd,1,local->inn,1);
free_real_matrix(local->wt_ef,1,local->nc, 1);
free_real_matrix(local->d_F,1,3,1);
free_real_matrix(local->d_Fc,1,3,1);
free_real_vector(local->d,1);

  kprintf("cfa: OFF\n");
  printf("Stopping the Contact Force Assignment Module \n");
  return I_OK;
}


/* ********************************************************************** */
/* cfaKill Clean up after the module. */
/* ********************************************************************** */

int cfaKill(local, stask)
cfaLocal_t *local;
sbsTask_t *stask;
{
  kprintf("cfa: FINISHED\n");
  return I_OK;
}


/* ********************************************************************** */
/* cfaError Attempt automatic error recovery. */
```

```
/* ********************************************************************** */

int cfaError(local, stask, mptr, errmsg, errcode)
cfaLocal_t *local;
sbsTask_t *stask;
errModule_t *mptr;
char *errmsg;
int errcode;
{
  /* Return after not correcting error. */

  return SBS_ERROR;
}



/* ********************************************************************** */
/* cfaClear Clear error state of the module. */
/* ********************************************************************** */

int cfaClear(local, stask, mptr, errmsg, errcode)
cfaLocal_t *local;
sbsTask_t *stask;
errModule_t *mptr;
char *errmsg;
int errcode;
{
  /* Return after not clearing error. */

  sbsNewError(stask, "Clear not defined, still in error state", errcode);
  return SBS_ERROR;
```

```
}


/* ********************************************************************** */
/* cfaSet Set module parameters. */
/* ********************************************************************** */

int cfaSet(local, stask)
cfaLocal_t *local;
sbsTask_t *stask;
{
  return I_OK;
}


/* ********************************************************************** */
/* cfaGet Get module parameters. */
/* ********************************************************************** */

int cfaGet(local, stask)
cfaLocal_t *local;
sbsTask_t *stask;
{
  return I_OK;
}


/* ********************************************************************** */
/* cfaSync For modules which synchronize to */
/* something other than the clock. */
```

```
/* ********************************************************************** */

int cfaSync(local, stask)
cfaLocal_t *local;
sbsTask_t *stask;
{
  return I_OK;
}

void h_sol( float mu, float del_X, float fc_x, int n, int num_I, int inn,
int **RB_ef, float c_dot[], int inummem,
float cfd_wt[], float afd_wt[],
float **T_i, float **e_I, float **e_Ic,
float **Fo, float **F, float Q[], float **Wp,
float **F_i_old, float **Fc,
float *theta, float *fc_check, float **ER_f, float **ER_fc,
float *v1, float *ER_dot, float *wt_tot, int **imf_ERd,
float **mu_ERd, float **wt_ef, float *nER_f, float **d_F,
float **d_Fc, float *d)

{

static char fname[80];
static char title[80];
   static int i,j,k,l,jj,ii,im1n,jj1,im1n1,im1n2,im1n3,I_count,i_wt_mf,nr_flag;
static float sum, sum1, ERdotmax, fabsERdot, C;
static float a,b,c,dp,dn;
static float mu2, fc_x2,d_X,dX,dx;
static int i_error;
```

```
/* initialize vectors and matrices before starting the solution. */

inimat(1,3,1,n,F_i_old,0);
inimat(1,3,1,n,Fc,0);
inivec(1,n,theta,0);
inivec(1,n,fc_check,0);
inimat(1,3,1,n,ER_f,0);
inimat(1,3,1,n,ER_fc,0);
inivec(1,3,v1,0);
inivec(1,inn,ER_dot,0);
inivec(1,inn,wt_tot,0);
inivec(1,n,nER_f,0);
ini_imat(1,inn,1,2,imf_ERd,0);
inimat(1,inn,1,2,mu_ERd,0);
inimat(1,inummem,1,num_I,wt_ef,0);
inimat(1,3,1,n,d_F,0);
inimat(1,3,1,n,d_Fc,0);
inivec(1,inn,d,0);


mu2=mu*mu;
    fc_x2=fc_x*fc_x;

/* del_X is the convergence buffer, fc_x is the stability offset */
dX=del_X+fc_x;


/* calculate external contact forces based on psuedoinverse and object wrench */

for(i=1;i<=n;i++)
```

```
{
for(l=1;l<=3;l++)
       {
          sum=0;
          for(j=1;j<=6;j++)
{
sum=sum+Wp[(i-1)*3+l][j]*Q[j];
}
Fo[l][i]=sum;
}
    }

/* calculate current contact forces */
for(j=1;j<=n;j++)
{
    for(i=1;i<=3;i++)
       {
F[i][j]=Fo[i][j]+F_i_old[i][j];
    }
}

   /* Fc=(T_i)'*F;  */
  /* convert each contact force into local contact frame coordinates */
for(i=1;i<=n;i++)
{
im1n=(i-1)*n;
for(j=1;j<=3;j++)
{

          jj=im1n+j;
```

```
          sum=0;
for(k=1;k<=3;k++)
{
sum=sum+T_i[k][jj]*F[k][i];
}
Fc[j][i]=sum;
}

/* theta is angle, in local y-z plane, to the external contact point */
theta[i]=atan2(Fc[3][i],Fc[2][i]);
}

/* check to see if any current contact forces */
/* are outside of the friction cones */
for(i=1;i<=n;i++)
{
fc_check[i]=(-mu2*(Fc[1][i]*Fc[1][i]-2*Fc[1][i]*fc_x)*
(-(Fc[1][i]<0)+(Fc[1][i]>=0)))-fc_x2*mu2+
(Fc[2][i]*Fc[2][i]+Fc[3][i]*Fc[3][i]);
}

/* begin iterative loop to find viable solution */

    i_error=0;
while(mymax(1,n,fc_check)>0)   /* contact stability violated */
{

i_error=i_error+1;

/************************************/
```

```
    /* print out variables to file fname */
/*strcpy(fname,"hs_out");*/
/*strcpy(title,"Contact Force, Contact Frame, Fc \n");*/
/*m_out(Fc, 1, 3, 1, n, title, fname);*/
/************************************/

/************************************/
   /* print out variables to file fname */
/*strcpy(fname,"hs_out");*/
/*strcpy(title,"fc_check \n");*/
/*v_out(fc_check, 1, n, title, fname);*/
/************************************/

/* determine what the goal vector should be, */
/* based on the direction of Fo */

for(i=1;i<=n;i++)
{
          im1n=(i-1)*n;
/* calculate error based on current F and goal */
if(fc_check[i]<=0) /*contact force is inside friction cone */
{
for(j=1;j<=3;j++)
             {
ER_fc[j][i]=0;
}
          }

/* move the friction cone to the inside of the nominal cone by dX */
/* along local x-axis; dX is used for solution convergence, not */
```

```
        /* contact stability, del_X is used for stability */

else  /* contact force is outside of friction cone */
{
dx=(Fc[1][i]+mu*(Fc[2][i]*cos(theta[i])+
Fc[3][i]*sin(theta[i])+mu*dX))/(1+mu2);

dx=dx*(dx>=dX)+dX*(dx<dX);

ER_fc[1][i]=dx-Fc[1][i];
ER_fc[2][i]=(dx-dX)*mu*cos(theta[i])-Fc[2][i];
ER_fc[3][i]=(dx-dX)*mu*sin(theta[i])-Fc[3][i];
}

/* convert errors in contact force, ER_fc, into object frame */
/* ER_f=T_i*ER_fc;  */

for(k=1;k<=3;k++)
{
       sum=0;
for(j=1;j<=3;j++)
{
jj=im1n+j;
sum=sum+T_i[k][jj]*ER_fc[j][i];
       }
ER_f[k][i]=sum;
       }

for(j=1;j<=3;j++)
{
```

```
v1[j]=ER_f[j][i];
          }
nER_f[i]=euclnorm(1,3,v1);
}

/************************************/
    /* print out variables to file fname */
/*strcpy(fname,"hs_out");*/
/*strcpy(title,"ER_f \n");*/
/*m_out(ER_f, 1, 3, 1,n, title, fname);*/
/************************************/

/************************************/
    /* print out variables to file fname */
/*strcpy(fname,"hs_out");*/
/*strcpy(title,"nER_f \n");*/
/*v_out(nER_f, 1, n, title, fname);*/
/************************************/

/* loop through all the internal forces */
       ERdotmax=0;
for(i=1;i<=n;i++)
{
im1n=(i-1)*n;

for(j=1;j<=n;j++)
{
jj=im1n+j;

            /* calculate the dot product between the contact error */
```

```
/* and the internal force for each contact */

            sum=0;
for(k=1;k<=3;k++)
{
sum=sum+e_Ic[k][jj]*ER_fc[k][i];
}
ER_dot[jj]=sum;

/* determine max value of ER_dot for normalization or ER_dot */
            fabsERdot=fabs(ER_dot[jj]);
if(fabsERdot > ERdotmax)
{
ERdotmax=fabsERdot;
            }
}
      }

/* can't divide by zero in normalization */
if(ERdotmax==0)
{
ERdotmax=1;
}

for(i=1;i<=inn;i++)
{
ER_dot[i]=ER_dot[i]/ERdotmax;
}

/************************************/
```

```
    /* print out variables to file fname */
/*strcpy(fname,"hs_out");*/
/*strcpy(title,"ER_dot \n");*/
/*v_out(ER_dot, 1,inn, title, fname);*/
/************************************/


/* calculate memberships of the ER_dot scalars */
mem_edci(imf_ERd, mu_ERd, ER_dot, inn, c_dot, inummem);

/************************************/
    /* print out variables to file fname */
/*strcpy(fname,"hs_out");*/
/*strcpy(title,"imf_ERd \n");*/
/*im_out(imf_ERd, 1,inn,1,2, title, fname);*/
/************************************/

/************************************/
    /* print out variables to file fname */
/*strcpy(fname,"hs_out");*/
/*strcpy(title,"mu_ERd \n");*/
/*m_out(mu_ERd, 1,inn,1,2, title, fname);*/
/************************************/

/* now apply the rule base to the input data in order to weight the */
/* internal forces */
inimat(1,inummem,1,num_I,wt_ef,0);
I_count=0;
for(i=1;i<=n;i++)
{
```

```
for(j=1;j<=n;j++)
{
if(i!=j && i<j)
{
jj=(i-1)*n+j;
ii=(j-1)*n+i;
I_count=I_count+1;

for(l=1;l<=2;l++)
{
for(k=1;k<=2;k++)
{
v1[1]=mu_ERd[jj][l];
v1[2]=mu_ERd[ii][k];

                    i_wt_mf=RB_ef[imf_ERd[jj][l]][imf_ERd[ii][k]];
wt_ef[i_wt_mf][I_count]=wt_ef[i_wt_mf][I_count]+mymin(1,2,v1);
}
}
}
}
}

/* at this point, 'I_count' should equal 'num_I'*/
/* which is the number of internal forces */

for(i=1;i<=num_I;i++)
{
sum=0;
sum1=0;
```

```
for(j=1;j<=inummem;j++)
{
sum=sum+afd_wt[j]*cfd_wt[j]*wt_ef[j][i];
sum1=sum1+afd_wt[j]*wt_ef[j][i];
}
wt_tot[i]=sum/sum1;
}

/************************************/
    /* print out variables to file fname */
/*strcpy(fname,"hs_out");*/
/*strcpy(title,"wt_tot \n");*/
/*v_out(wt_tot, 1, num_I, title, fname);*/
/************************************/
inimat(1,3,1,n,d_F,0);
I_count=0;
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
if(i!=j && i<j)
{
jj=(i-1)*n+j;
ii=(j-1)*n+i;
I_count=I_count+1;

for(k=1;k<=3;k++)
{
d_F[k][i]=d_F[k][i]+wt_tot[I_count]*e_I[k][jj];
d_F[k][j]=d_F[k][j]+wt_tot[I_count]*e_I[k][ii];
```

```
}
}
}
        }

/************************************/
    /* print out variables to file fname */
/*strcpy(fname,"hs_out");*/
/*strcpy(title,"d_F \n");*/
/*m_out(d_F, 1, 3, 1, n, title, fname);*/
/************************************/

/* determine contact, j, with most significant error, sum */
        sum=0;
for(i=1;i<=n;i++)
{
if(nER_f[i]>sum)
{
j=i;
sum=nER_f[i];
}
        }

/* determine the psuedoinverse of j'th column of d_F */
/* since v1 is a vector, the transpose of the psuedoinverse is: */
/* v1#t=v1 / (v1[1]^2 + v1[2]^2 + v1[3]^2)   */

        sum=0;
    for(i=1;i<=3;i++)
{
```

```
sum=sum+d_F[i][j]*d_F[i][j];
}

for(i=1;i<=3;i++)
{
v1[i]=d_F[i][j]/sum;
}

        /* v1 is now the transpose of the psuedoinverse of d_F[:][j] */

        C=0;
for(i=1;i<=3;i++)
{
C=C+v1[i]*ER_f[i][j];
}
/* C is the scale factor for the change in contact forces, d_F */

   /************************************/
    /* print out variables to file fname */
/*strcpy(fname,"hs_out");*/
/*strcpy(title,"C \n");*/
/*s_out(C, title, fname);*/
/************************************/

for(i=1;i<=n;i++)
{
for(j=1;j<=3;j++)
{
F_i_old[j][i]=F_i_old[j][i]+C*d_F[j][i];
}
```

```
        }

        /* update contact forces */
for(j=1;j<=n;j++)
{
    for(i=1;i<=3;i++)
      {
F[i][j]=Fo[i][j]+F_i_old[i][j];
    }
}

/************************************/
/* print out variables to file fname */
/*strcpy(fname,"hs_out");*/
/*strcpy(title,"F \n");*/
/*m_out(F, 1,3,1,n, title, fname);*/
/************************************/


   /* Fc=(T_i)'*F;  */
  /* convert each contact force into local contact frame coordinates */
for(i=1;i<=n;i++)
{
im1n=(i-1)*n;
for(j=1;j<=3;j++)
{
   jj=im1n+j;
        sum=0;
for(k=1;k<=3;k++)
{
```

```
sum=sum+T_i[k][jj]*F[k][i];
}
Fc[j][i]=sum;
}
}


/* check to see if any current contact forces */
/* are outside of the friction cones */
for(i=1;i<=n;i++)
{
if(i_error<15)
          {
fc_check[i]=(-mu2*(Fc[1][i]*Fc[1][i]-2*Fc[1][i]*fc_x)*
(-(Fc[1][i]<0)+(Fc[1][i]>=0)))-fc_x2*mu2+
(Fc[2][i]*Fc[2][i]+Fc[3][i]*Fc[3][i]);
}

else
{
fc_check[i]=-1;
/************************************/
/* print out variables to file fname */
/*strcpy(fname,"hs_out");*/
/*strcpy(title,"i_error, Over 15 Attempts at Solution Failed \n");*/
/*s_out(i_error, title, fname);*/
/************************************/
          }
}
```

```
} /* end while loop for contact stability */


} /* end h_sol */

float mymax(int l, int u, float b[])
/* this function takes a vector and returns the maximum value of the vector */
/* l and u are the lower and upper indices of the vector. */
{
int i;
float max;

    max=b[l];

for(i=l+1;i<=u;i++)
{
if(b[i]>max)
{
max=b[i];
}
}
return max;
} /* end max */


float mymin(int l, int u, float b[])
/* this function takes a vector and returns the minimum value of the vector */
/* l and u are the lower and upper indices of the vector. */
{
```

```
int i;
float min;

    min=b[1];

for(i=l+1;i<=u;i++)
{
if(b[i]<min)
{
min=b[i];
}
}
return min;
} /* end min */

void mem_edci( int **imf_ERd, float **mu_ERd, float ER_dot[],
int n, float c_dot[], int m)
{
/* subroutine which calculates the membership of a given input vector    */
/* with respect to the triangular fuzzy membership functions defined  */
/* by the vector of 'centers.'  Use this version of membership calc-      */
/* ulation for external force, dot product of internal and external forces,*/
/* cross product of internal force and friction cone vectors, and output */
/* weights associated with each internal force.                          */
/* Mark Hunter, 20 Nov 95, Air Force Institute of Technology, WPAFB, OH. */

/* mf is the membership function label of the first non-zero membership  */
/* mu(i,1) is the value of the membership of x(i) with repect to mf(i)   */
/* index(i) refers to the number of membership functions crossed by x(i) */
/* n is the number of contacts squared */
```

```
/* m is the number of membership functions across the domain */

int i, j, j_flag;

for(i=1;i<=n;i++)
{
j=0;
j_flag=0;

if(ER_dot[i]<=c_dot[1])
{
j_flag=1;

imf_ERd[i][1]=1;
imf_ERd[i][2]=2;
mu_ERd[i][1]=1;
mu_ERd[i][2]=0;
}
else if(ER_dot[i]>=c_dot[m])
{
j_flag=1;

imf_ERd[i][1]=m-1;
imf_ERd[i][2]=m;
mu_ERd[i][1]=0;
mu_ERd[i][2]=1;
}

       while(j_flag==0)
{
```

```
j=j+1;

if(ER_dot[i]<c_dot[j+1])
{
j_flag=1;

  imf_ERd[i][1]=j;
imf_ERd[i][2]=j+1;
mu_ERd[i][1]=(c_dot[j+1] - ER_dot[i])/
(c_dot[j+1] - c_dot[j]);
mu_ERd[i][2]=(ER_dot[i] - c_dot[j])/
(c_dot[j+1] - c_dot[j]);
}



} /* end while */

} /* end for */

}  /* end mem_edci */

void cross(int l, int u, float v1[], float v2[], float v3[])
{
int m=u-l;

v3[l]=v1[m]*v2[u]-v2[m]*v1[u];
v3[m]=v1[u]*v2[l]-v2[u]*v1[l];
v3[u]=v1[l]*v2[m]-v2[l]*v1[m];
}
```

```
float euclnorm(int l, int u, float b[])
/* calculate euclidean norm of vector a[l:u] */
{
   int i;
float enorm=0.0;

for (i=l; i<=u; i++)
{
enorm = enorm + b[i]*b[i];
}

return sqrt(enorm);
}

void m_out(float **a, int rl, int ru, int cl,
int cu, char title[], char file[])
/* print matrix value to file */
{
 char fn[8];
 int i,j;
 FILE *fp;

 for (i=0; i<8; i++)
{
fn[i]=file[i];
}

 fp = fopen (fn,"a");

 fprintf(fp,"%s",title);
```

```
 for (i=rl;i<=ru;i++)
 {
fprintf(fp,"\n");
for (j=cl;j<=cu;j++)
{
fprintf(fp," %12.4e ", a[i][j]);
}
 }
 fprintf(fp,"\n");

 fclose(fp);
}

void im_out(int **a, int rl, int ru,
int cl, int cu, char title[], char file[])
/* print matrix value to file */
{
 char fn[8];
 int i,j;
 FILE *fp;

 for (i=0; i<8; i++)
{
fn[i]=file[i];
}

 fp = fopen (fn,"a");

 fprintf(fp,"%s",title);
```

```
 for (i=rl;i<=ru;i++)
 {
fprintf(fp,"\n");
for (j=cl;j<=cu;j++)
{
fprintf(fp," %d ", a[i][j]);
}
 }
 fprintf(fp,"\n");

 fclose(fp);
}

void v_out(float b[], int l, int u,
char title[], char file[])
/* print vector value to file */
{
 char fn[8];
 int i;
 FILE *fp;

 for (i=0; i<8; i++)
{
fn[i]=file[i];
}

 fp = fopen (fn,"a");

 fprintf(fp, "%s",title);
```

```
 for (i=l;i<=u;i++)
 {
fprintf(fp," %12.4e \n ",b[i]);
 }
 fprintf(fp,"\n");

 fclose(fp);
}

void s_out(float c, char title[], char file[])
/* print scalar value to file */
{
 char fn[8];
 int i;

 FILE *fp;

 for (i=0; i<8; i++)
 {
fn[i]=file[i];
 }

 fp = fopen (fn,"a");

 fprintf(fp, "%s",title);

 fprintf(fp," %12.4e \n ",c);

 fprintf(fp,"\n");
```

```
 fclose(fp);

}
```

Bibliography

1. *The MATLAB Reference Guide*. 24 Prime Park Way, Natick, MA 01760: The Math Works Inc., 1992.

2. *The SIMULINK User's Guide*. 24 Prime Park Way, Natick, MA 01760: The Math Works Inc., 1992.

3. Ahmad, S. "Control of Cooperative Multiple Flexible Joint Robots," *IEEE Transactions on Systems, Man, and Cybernetics*, *23*(3):833–839 (May/June 1993).

4. Alberts, T.E. and D.I. Soloway. "Force Control of a Multi-Arm Robot System." *Proceedings of the IEEE Conference on Robotics and Automation*. 1490–1496. 1988.

5. Apostol, T.M. *Mathematical Analysis*. Menlo Park, CA: Addison-Wesley Publishing Co., 1974.

6. Bobrow, J.E., J.M. McCarthy and V.K. Chu. "Time-Optimal Control of Two Robots Holding the Same Workpiece," *ASME J. of Dynamic Systems, Measurement, and Control*, *115*:441–446 (September 1993).

7. Bonitz, R.G. and T.C. Hsia. "Force Decomposition in Cooperating Manipulators Using the Theory of Metric Spaces and Generalized Inverses." *Proceedings of the IEEE International Conference on Robotics and Automation*. 1521–1527. 1994.

8. Bonitz, R.G. and T.C. Hsia. "Robust Internal Force-Based Impedance Control for Cooperating Manipulators-Theory and Experiments." *Proceedings of the IEEE International Conference on Robotics and Automation*. 622–628. 1996.

9. Buss, M., H. Hashimoto and J.B. Moore. "Grasping Force Optimization for Multi-Fingered Robot Hands." *Proceedings of the IEEE International Conference on Robotics and Automation*. 1034–1039. 1995.

10. Chen, Y.-C., I.D. Walker and J.B. Cheatham. "A New Approach to Force Distribution and Planning for Multifingered Grasps of Solid Objects." *Proceedings of the IEEE Conference on Robotics and Automation*. 890–896. 1991.

11. Conner, D. "Fuzzy-Logic Control System," *EDN*, 76–88 (March 1993).

12. Cox, E. "Adaptive Fuzzy Systems," *IEEE Spectrum*, *30*(2):27–31 (February 1993).

13. Cox, E. *Fuzzy systems handbook*. New York: AP Professional, 1994.

14. Crago, P.E., R.J. Nakai and H.J. Chizeck. "Feedback Regulation of Hand Grasp Opening and Contact Force During Stimulation of Paralyzed Muscle," *IEEE Transactions on Biomedical Engineering*, *38*(1):17–28 (January 1991).

15. Cutkosky, M. R. and P. K. Wright. "Friction, stability and the design of robotic fingers," *Int. Journal of Robotics Research*, *5*(4):20–37 (1986).

16. Fagg, A.H., D. Lotspeich and G.A. Bekey. "A Reinforcement-Learning Approach to Reactive Control Policy Design for Autonomous Robots." *Proceeding of the IEEE International Conference on Robotics and Automation*. 39–44. 1994.

17. Fu, K.S., R.C. Gonzalez and C.S.G. Lee. *Robotics: Control, Sensing, Vision, and Intelligence*. New York: McGraw-Hill Book Co., 1984.

18. Giarratona, J.C. and G. Riley. *Expert systems, principles and programming*. Boston: PWS-Kent Publishing Company, 1989.

19. Grace, A. *Optimization TOOLBOX*. 24 Prime Park Way, Natick, MA 01760: The Math Works Inc., 1992.

20. Hogan, N. "Impedance control: an approach to manipulation," *ASME J. of Dynamic Systems, Measurement, and Control*, *107*:1–23 (March 1985).

21. Hollerbach, J. M., S. Narasimhan and J. E. Wood. "Finger Force Computation without the Grip Jacobian." *Proceedings of the IEEE Conference on Robotics and Automation*. 871–875. 1986.

22. Holzmann, W. and J.M. McCarthy. "Computing the Friction Forces Associated with a Three-Fingered Grasp," *IEEE Journal of Robotics and Automation*, *1*(4):206–210 (Dec 1985).

23. Iberall, T., G.S. Sukhatme D. Beattie and G.A. Bekey. "On the Development of EMG Control for a Prosthesis Using a Robotic Hand." *Proceedings of the IEEE Conference on Robotics and Automation*. 1753–1758. 1994.

24. Ji, Z. and B. Roth. "Direct Computation of Grasping Force for Three-Finger Tip-Prehension Grasps," *ASME J. of Mechanisms, Transmissions, and Automation in Design*, *110*:405–413 (December 1988).

25. Kerr, J. and B. Roth. "Analysis of Multifingered Hands," *Int. J. Robotics Res.*, *4*(4):3–17 (Winter 1986).

26. Klafter, R.D., T.A. Chmielewski and M. Negin. *Robotic Engineering: An Integrated Approach*. Englewood Cliffs, New Jersey: Prentice Hall, 1989.

27. Kobayashi, H. "Control and Geometrical Considerations for an Articulated Robot Hand," *Int. J. Robotics Res.*, *4*(1):3–12 (Spring 1985).

28. Kosko, B. *Neural Networks and Fuzzy Systems*. Englewood Cliffs: Prentice Hall, 1992.

29. Kosko, B. *Fuzzy thinking*. New York: Hyperion, 1993.

30. Kumar, V. and K.J. Waldron. "Sub-Optimal Algorithms for Force Distribution in Multifingered Grippers." *Proceedings of the IEEE Conference on Robotics and Automation*. 2992–2999. 1987.

31. Kwong, W.A., K.M. Passino and S. Yurkovich. "Fuzzy Learning Systems for Aircraft Control Law Reconfiguration." *IEEE International Symposium on Intelligent Control.* 333–338. 1994.

32. Li, L., P. Hsu and S. Sastry. "Grasping and Coordinated Manipulation by a Multifingered Robot Hand," *Int. J. Robotics Res.*, *8*(4):33–50 (August 1989).

33. Mason, M.T. and Jr. J.K. Salisbury. *Robot Hands and the Mechanics of Manipulation.* Cambridge: The MIT Press, 1985.

34. Michelman, P. and P. Allen. "Forming Complex Dextrous Manipulations from Task Primitives." *Proceedings of the IEEE Conference on Robotics and Automation.* 3383–3388. 1994.

35. Murray, R.M. and S.S. Sastry. "Control Experiments in Planar Manipulation and Grasping." *Proceedings of the IEEE Conference on Robotics and Automation.* 624–629. 1989.

36. Murray, R.M., Z. Li and S.S. Sastry. *A Mathematical Introduction to Robotic Manipulation.* Boca Raton: CRC Press, Inc., 1994.

37. Nagai, K. and T. Yoshikawa. "Dynamic Manipulation/Grasping Control of Multifingered Robot Hands." *Proceedings of the IEEE Conference on Robotics and Automation.* 1027–1033. 1993.

38. Nakamura, Y., K. Nagai and T. Yoshikawa. "Mechanics of Coordinative Manipulation by Multiple Robotic Mechanisms." *Proceedings of the IEEE Conference on Robotics and Automation.* 991–998. 1987.

39. Nakamura, Y., K. Nagai and T. Yoshikawa. "Dynamics and Stability in Coordination of Multiple Robotic Mechanisms," *Int. J. Robotics Res.*, *8*(2):44–61 (April 1989).

40. Nakamura, Y. *Advanced Robotics, Redundancy and Optimization.* New York: Addison-Wesley, 1991.

41. Napier, J. "The Prehensile Movements fo the Human Hand," *The Journal of Bone and Joint Surgery*, *38B*(4):902–913 (Nov 1956).

42. Orin, D.E. and S.Y. Oh. "Control of Force Distribution in Robotic Mechanisms Containing Closed Kinematic Chains," *ASME J. of Dynamic Systems, Measurement, and Control*, *102*:134–141 (June 1981).

43. Park, Y.C. and G.P. Starr. "Finger Force Computation for Manipulation of an Object be a Multifingered Robot Hand." *Proceedings of the IEEE Conference on Robotics and Automation.* 930–935. 1989.

44. Raibert, M.H. and J.J. Craig. "Hybrid position/force control of manipulators," *ASME J. of Dynamic Systems, Measurement, and Control*, *102*:126–133 (June 1981).

45. Rattan, K.S., T. Brehm and G.S. Sandhu. "Analysis and Design of a Proportional Fuzzy Logic Controller." *Proceedings of the International Fuzzy Systems and Intelligent Control Conference*. 17–26. 1994.

46. Repperger, D.W., T.L. Chelette and C.A. Phillips. "Identification of Membership Function Parameters with Empirical Data from a Biomedical Application." *Proceedings of the IEEE International Symposium on Intelligent Control*. 309–314. 1994.

47. Salisbury, J.K. and B. Roth. "Kinematic and Force Analysis of Articulated Mechanical Hands," *ASME J. of Mechanisms, Transmissions, and Automation in Design*, *105*:35–41 (March 1983).

48. Sandhu, G.S., K.S. Rattan and T. Brehm. "Analysis and Design of a Proportional Plus Derivative Fuzzy Logic Controller." *Proceedings of the NAECON*. –. 1996.

49. Schneider, S.A. and R.H. Cannon Jr. "Object Impedance Control for Cooperative Manipulation: Theory and Experimental Results." *Proceedings of the IEEE Conference on Robotics and Automation*. 1076–1083. 1989.

50. Sinha, P.R. and J.M. Abel. "Manipulating and Grasping Forces in Manipulation by Multifingered Robot Hands," *IEEE Transactions on Robotics and Automation*, *8*(1):67–77 (February 1992).

51. Sugeno, M. *Idustrial applications of fuzzy control*. New York: Elsevier Science Publishing Co., 1985.

52. T.J., Ross. *Fuzzy Logic with Engineering Applications*. New York, New York: McGraw-Hill, Inc., 1995.

53. Vachtsevanos, G., S.S. Farinwata and D.K. Pirovolou. "Fuzzy Logic Control of an Automotive Engine," *IEEE Control Systems*, 62–68 (June 1993).

54. Wallich, P. "Silicon Babies," *Scientific American*, 124–134 (December 1991).

55. Wang, L.-X. and J.M. Mendel. "Generating fuzzy rules by learning from examples." *Proceedings of the IEEE International Symposium on Intelligent Control*. 263–268. 1991.

56. Wen, J.T. and K. Kreutz-Delgado. "Motion and Force Control of Multiple Robotic Manipulators," *Automatica*, *28*(4):729–743 (1992).

57. Yager, R.R. and D.P. Filev. *Essentials of fuzzy modeling and control*. New York: John Wiley & Sons, Inc., 1994.

58. Yager, R.R. and L.A. Zadeh. *Fuzzy Sets, Neural Networks, and Soft Computing*. New York: Van Nostrand Reinhold, 1994.

59. Yao, B. and M. Tomizuka. "Adaptive Coordinated Control of Multiple Manipulators Handling a Constrained Object." *Proceeding of the IEEE International Conference of Robotics and Automation*. 624–629. 1993.

60. Yoshikawa, T. "Manipulability of Robotic Mechanisms," *Int. J. Robotics Res.*, *4*(2):3–9 (Summer 1985).

61. Yoshikawa, T. "Dynamic Hybrid Position/Force Control of Robot Manipulators," *IEEE J. of Robotics and Automation*, *3*(5):386–392 (October 1987).

62. Yoshikawa, T. "Passive and Active Closures by Constraining Mechanisms." *Proceeding of the IEEE International Conference of Robotics and Automation.* 1477–1484. 1996.

63. Yoshikawa, T. and K. Nagai. "Manipulating and Grasping Forces in Manipulation by Multifingered Robot Hands," *IEEE Transactions on Robotics and Automation*, 7(1):67–77 (February 1991).

64. Zimmermann, H.-J. *Fuzzy Set Theory and Its Applications.* Boston: Kluwer Academic Publishers, 1991.

Vita

Captain Mark W. Hunter [redacted] He graduated from Carmel High School in Carmel, California and attended Virginia Polytechnic Institute and State University, graduating with a Bachelor of Science in Aerospace Engineering in July 1985. Upon graduation, he attended Officer Training School at Lackland AFB, Texas where he was a distinguished graduate. His first assignment as a Second Lieutenant, was to Tinker AFB, Oklahoma. He served as a B-1B engineer during test and evaluation efforts and field level modifications. He attended the Air Force Institute of Technology, Wright-Patterson AFB, Ohio from May 1989 until December 1990 graduating with a Master of Science in Aeronautical Engineering. He then served as an aeronautical engineer with the Aircraft Systems Center, Wright-Patterson AFB, Ohio developing innovative computer codes for aerodynamic analysis of conceptual aircraft designs. He again attended the Air Force Institute of Technology from July 1993 until July 1996.

[redacted]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1996	Doctoral Dissertation

4. TITLE AND SUBTITLE
REACTION BASED GRASP FORCE ASSIGNMENT

5. FUNDING NUMBERS

6. AUTHOR(S)
Mark W. Hunter, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
2750 P Street
WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/DS/ENY/96-10

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
SA-ALC/TIEST, Bldg 183
450 Quentin Roosevelt Rd
Kelly AFB, TX 78241-6416

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved For Public Release;
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

An iterative method is developed by which the contact forces required to apply an arbitrary wrench (six elements of force and moment) to a stably grasped object may be calculated quickly. The assignment of contact forces, given a required object wrench, is accomplished with the use of fuzzy logic. This concept is referred to as the fuzzy logic reactive system (FLRS). The solution is versatile with respect to goals inherent in the rulebase and the input parameters, and is also applicable for an arbitrary number of contacts. The goal presented in this research, to illustrate the concept of the FLRS, is the minimization of the norm of the contact forces using point contacts with friction. Results comparing the contact force assignment for this method and the optimal method proposed by Nakamura are presented. The results show that FLRS will satisfy the object wrench and frictional contacts while achieving near optimal contact force assignment. This method is shown to require significantly fewer floating point operations than the solution calculated using numerical constrained optimization techniques.

14. SUBJECT TERMS
Robotic Grasping, Contact Force, Multirobot Coordination, Object Manipulation

15. NUMBER OF PAGES
227

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL